NASA Conference Publication 2478

Frontiers of Massively Parallel Scientific Computation

James R. Fishcher, *Editor*
Goddard Space Flight Center
Greenbelt, Maryland

Proceedings of the first symposium sponsored by the
National Aeronautics and Space Administration,
Washington, D.C., and the Goodyear Aerospace
Corporation, Akron, Ohio, and held at
NASA Goddard Space Flight Center
Greenbelt, Maryland
September 24–25, 1986

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Office

1987

PREFACE

Practical scientific applications using massively parallel computer hardware first appeared on the research scene during the 1980s. The development of these applications was motivated by the need for computing power orders of magnitude beyond what is available today for large-scale tasks such as numerical simulations of complex physical and biological processes, generation of interactive visual displays, satellite image analysis, and knowledge-based systems.

The *First Symposium on the Frontiers of Massively Parallel Scientific Computation,* held September 24-25, 1986, at NASA's Goddard Space Flight Center in Greenbelt, Maryland, provided a glimpse of the emerging power of massively parallel computers in solving a wide range of scientific problems. This proceedings is a compilation of all papers given during the 2-day meeting.

Sponsored by NASA and Goodyear Aerospace Corporation, the symposium featured 33 speakers representing 11 universities, 3 research institutions, 7 companies, and 5 government laboratories. The papers were grouped into five subject areas: Earth sciences, physics, computer science, signal and image processing, and computer graphics. More than 250 people attended the symposium, representing 40 corporate and industrial research laboratories, 29 universities, 13 government agencies, and 5 NASA centers.

We are very pleased to have brought together--for the first time--applications papers on three massively parallel computers: the Distributed Array Processor (DAP) (one invited paper), the Connection Machine (two invited papers), and the Massively Parallel Processor (MPP) (28 papers from the MPP Working Group). Three additional papers addressed the next generation of massively parallel hardware and current developments in their system software.

The diversity of applications running on massively parallel hardware illustrates the general purpose capabilities of this architecture. The DAP--in service in England for 8 years--produced a wealth of the "new intuition" necessary for its effective use. The commercially available Connection Machine is being developed for artificial intelligence applications. The MPP, though designed to perform image processing, is performing numerical simulations for a majority of its applications.

Massively parallel computers have emerged as inexpensive alternatives to other supercomputers. And, because their architectures can be upgraded fairly easily, compelling scientific and marketplace arguments are made for accelerating their development.

As parallel processing continues to gain ground in the scientific arena, we look forward to continuing these *Frontiers* symposia, presenting the latest in scientific parallel algorithms.

James R. Fischer
Symposium Committee Chairman, and

Dr. James C. Tilton
Program Chairman

PRECEDING PAGE BLANK NOT FILMED

FOREWORD

As NASA moves into the Space Station era with its permanent space-based observatories, it is faced with processing, analyzing, and storing a monumental volume of spectrally acquired image data. A few years ago, in anticipation of these future flight systems, NASA initiated a technology development effort to exploit parallel computation architectures for dealing with the high-resolution image data. This technology effort developed the Massively Parallel Processor (MPP), which has recently reached operational status.

Once the MPP was available, NASA enlisted the assistance of the broader science community by offering them the opportunity to explore its capabilities. Through a Space Science and Applications Notice issued nationwide in December 1984, NASA selected computational investigations for implementation on the MPP. NASA formed the principal investigators into a working group to evaluate and report on the MPP's performance and utility for scientific applications. These applications relate not just to space and Earth science problems, but also to a wide range of general physical and mathematical sciences.

The NASA-sponsored *First Symposium on the Frontiers of Massively Parallel Scientific Computation* provided a forum for the MPP Working Group members to share the results and experiences gained from their first year's scientific research in computer science, graphics, image processing, physics, and Earth sciences. Presentations, however, were not restricted to the MPP Working Group, and the symposium provided an opportunity to learn of computational applications on other massively parallel computers, including the Distributed Array Processor (DAP) and the Connection Machine.

The Information Systems Office was extremely pleased with the findings presented at this Symposium. The reported results provide very useful information about current and future capabilities of massively parallel computers to handle a wide variety of NASA problems. It was especially gratifying to learn of the wide range of applicability that parallel processing offers to the NASA Space and Earth Science Community. This was further amplified by the rapid start-up time needed to get scientific problems mapped and running on this new computer's architecture. These results justify NASA's earlier research and development support of this type of technology and are encouraging for continuation of the process of massively parallel scientific computational research in support of space and Earth science programs.

Dr. Caldwell McCoy, Jr.
Manager, Information Systems Office
NASA Headquarters

PRECEDING PAGE BLANK NOT FILMED

Acknowledgments

Many organizations and individuals made significant contributions to the success of the *First Symposium on the Frontiers of Massively Parallel Scientific Computation.* In particular:

the **MPP Working Group members,** along with their **technical staffs,** who committed their own valuable time over the past 2 years to explore the potential of the MPP in unique areas of research;

the **Invited Speakers,** who all immediately agreed to come, many on short notice, to broaden our program by sharing their research;

the **Session Chairmen;**

the **MPP Consultants,** who, beginning October 1, 1985, provided the Working Group members with whatever assistance was needed to achieve success with their MPP application by Symposium time;

Jim Abeles, Tim Busse, Judy Devaney, Bill Haynes, Ed Seiler, and **Dave Wildenhain** (Science Applications Research SAR); **Jim Hurst** and **Tor Opsahl** (formerly from SAR); **John Dorband, Jim Strong,** and **Jim Tilton** (NASA/GFSC);

the **Information Systems Office,** NASA Headquaraters, which supports operation of the MPP and the Working Group---in particular, Dr. Caldwell McCoy, manager, and Mrs. Ai Fang, program manager;

the **Office of Aeronautics and Space Technology,** NASA Headquarters, which supported the MPP development and demonstration from the beginning;

Goodyear Aerospace Corporation, whose co-sponsorship of the Symposium is indicative of a long and close association with the Goddard MPP program, beginning with the development of the MPP under NASA contract and meeting all its performance specifications--especially **Ken Batcher,** for the MPP design, **Rudi Faiss,** for developing the MPP transcendental function package, and **Doug Smith,** for his dedicated support of the MPP since delivery;

Science Applications Research (SAR), which provides systems, applications, and user support for the MPP, managed by **Ernest Daddio** (formerly by **Doug Sakal**), and who provided technical writing support, including the creation of our symposium poster and program;

the MPP User Support Office: **Emma McKnight** (SAR), who operates it, and **Dolores Parker** (SAR), who first organized it;

Carol Hoxie, assisted by **Mark Stevens** (SAR), who prepared this Proceedings and contributed substantially to every other aspect of the Symposium from planning to execution;

Jim Tilton, Program Chairman, who organized the Symposium's technical sessions, and who, along with **Ed Seiler, Jim Hurst, Jim Abeles,** and **Tor Opsahl,** located software landmines over the past 3 years, serving as guinea pigs for all improvements to the MPP languages and operating system;

George R. Rumney, II, who led the development of the MPP operating system, assisted by **Laura Potler, Lisa Hamet,** and **Jim Fry**. George and Jim provided extensive troubleshooting assistance to Working Group members and the MPP consultants;

Charles Cosner and **Sol Broder,** for making possible the network paths from the Working Group members' remote locations to the MPP;

at the University of Illinois: **Greg Smith, Andy Spry,** and **Dave Patton,** who wrote the MPP assemblers, and **Scott McEwan,** who wrote the MPP simulator;

Tony Reeves, Cornell University, who designed Parallel Pascal, and **Denis Lynch, Kevin Ingram,** and **Byron Leas** (CSC, Inc.), who implemented much of it for the MPP; also **Tim Busse** and **Jenny Wu** (SAR), who developed the new MPP Pascal;

PRECEDING PAGE BLANK NOT FILMED

Arnie Oakes, who provided logistical support during the process of selecting all Working Group members, and **Carol Arkwright** who, along with Arnie, assisted in planning the Working Group conferences;

Ernie Neil, who always found a way to get things done--always!;

Gary Wolford, Bob Grigsby, Millie Brice, and **Chuck Bohn** in the Image Analysis Facility, for Symposium logistical support;

Marie Stubbs, the Goddard International Coordinator, whose cool ability to get most folks through the front gate is highly valued;

Milt Halem, who recognized the latent potential of the MPP and accepted a personal challenge to develop it through the Working Group;

Dave Schaefer and **Jim Strong**, who dreamed up the concept for the MPP 16 years ago, and who keep getting new ideas;

and all the other visionaries who helped the MPP become a reality.

---Jim Fischer

CONTENTS

Preface iii
Foreword v
Acknowledgments vii

EARTH SCIENCES SESSION I

Session Chair: Dr. Robert J. Gurney
NASA Goddard Space Flight Center

An Implementation of a Barotropic Quasigeostrophic Model of Ocean Circulation on the MPP
Dr. Chester E. Grosch and
R. Fatoohi, Old Dominion University 3

Comparison of the MPP with Other Supercomputers for Landsat Data Processing
Martin Ozga, U.S. Department of Agriculture 13

Programming a Hillslope Water Movement Model on the MPP
Dr. Robert J. Gurney, and P. J. Camillo, NASA GFSC
J. E. Devaney, and A. R. Irving, Science Applications Research 19

PHYSICS SESSION I

Session Chair: Dr. L. R. Owen Storey
Stanford University

Phase Space Simulation of Collisionless Stellar Systems on the Massively Parallel Processor
Dr. Richard L. White, Space Telescope Science Institute 29

Particle Simulation of Plasmas on the Massively Parallel Processor
Dr. I. M. A. Gledhill and
Dr. L. R. Owen Storey, Stanford University 37

A Study of the XY Model by the Monte Carlo Method
Dr. Peter Suranyi and
Paul Harten, University of Cincinnati 47

A Fast MPP Algorithm for Ising Spin Exchange Simulations
Dr. Francis Sullivan and
Dr. Raymond Mountain, National Bureau of Standards 53

COMPUTER SCIENCE SESSION I

Session Chair: Dr. Michael A. McAnulty
University of Alabama in Birmingham

Simulation of an Array-Based Neural Net Model
Dr. John A. Barnden, Indiana University 59

Neural Nets on the MPP
Dr. Harold M. Hastings and
Dr. Stefan Waner, Hofstra University 69

Parallel VLSI Architecture Emulation and the Organization of APSA/MPP
Dr. John T. O'Donnell, Indiana University 75

INVITED PAPER

Cellular Automaton Supercomputing
Dr. Stephen Wolfram, Center for Complex Systems Research 87

SIGNAL & IMAGE PROCESSING SESSION

Session Chair: Dr. Edward W. Davis
North Carolina State University

Block Iterative Restoration of Astronomical Images with the Massively Parallel Processor
Dr. Sara R. Heap, NASA GSFC
Donald L. Lindler, Advanced Computer Concepts 99

Reconstruction of Coded Aperture Images
Dr. Michael J. Bielefeld, Computer Sciences Corporation
Dr. Lo I. Yin, NASA GSFC 111

Synthetic Aperture Radar Signal Processing on the MPP
Dr. H. K. Ramapriyan, NASA GSFC
Edward J. Seiler, Science Applications Research 121

An Application of the MPP to the Interactive Manipulation of Stereo Images of Digital Terrain Models
Dr. Edward W. Davis, Sanjay Pol, and Dr. David McAllister, 135
North Carolina State University

Development of a Stereo Analysis Algorithm for Generating Topographic Maps Using Interactive Techniques on the MPP
Dr. James P. Strong, NASA GSFC 141

AFTERDINNER KEYNOTE SPEAKER

The Architecture of Tomorrow's Massively Parallel Computer
Dr. Ken Batcher, Goodyear Aerospace Corporation 151

EARTH SCIENCES SESSION II

Session Chair: Dr. Chester E. Grosch
ICASE, NASA Langley Research Center

Implementation of the Barotropic Vorticity Equation on the MPP
Dr. Max J. Suarez, NASA GSFC, and
Jim Abeles, Science Applications Research 161

Chemical Network Problems Solved on NASA/Goddard's Massively Parallel Processor Computer
Dr. Gregory R. Carmichael, Seog Y. Cho, David M. Cohen, and Mehmet H. Oguztuzun, University of Iowa 165

Contextual Classification on the Massively Parallel Processor
Dr. James C. Tilton, NASA GSFC 171

PHYSICS SESSION II

Session Chair: Dr. Richard L. White
Space Telescope Science Institute

Plasma Simulation Using the Massively Parallel Processor
Dr. Chin S. Lin, A. L. Thring, J. Koga, and R.W. Janetzke, Southwest Research Institute 185

Free-Electron Laser Simulations on the MPP
Dr. Scott A. Von Laven, KMS Fusion
Lorie M. Liebrock, Michigan Technological University 193

Simulation of Charged Particle Transport on the MPP
Dr. James A. Earl, University of Maryland 197

INVITED PAPER

Experience in Highly Parallel Processing Using DAP
Dr. Dennis Parkinson, Queen Mary College 205

COMPUTER GRAPHICS SESSION

Session Chair: Dr. John A. Barnden
Indiana University

Ray Tracing on the MPP
Dr. John E. Dorband, NASA GSFC 211

Animated Computer Graphics Models of Space and Earth Sciences Data Generated via the Massively Parallel Processor
Lloyd A. Treinish, NASA GSFC
Michael L. Gough, and David W. Wildenhain, Science Applications Research 217

Real-Time Animation of Space Plasma Phenomena
Karen F. Jordan and
Eugene W. Greenstadt, TRW 223

Algorithmic Commonalities in the Parallel Environment
Dr. Michael A. McAnulty, and Michael S. Wainer, University of Alabama in Birmingham 227

INVITED PAPER

Reverse Time Migration---A Seismic Processing Application on the Connection Machine
Dr. Rolf D. Fiebrich, Thinking Machines Corporation 239

COMPUTER SCIENCE SESSION II

Session Chair: Dr. Harold M. Hastings
Hofstra University

Solving Very Large, Sparse Linear Systems on Mesh-Connected Parallel Computers
Dr. Torstein Opsahl, MRJ, Inc. 249
Dr. John H. Reif, Duke University

Solution of Large Linear Systems of Equations on the Massively Parallel Processor
Dr. Nathan Ida and
Kapila Udawatta, University of Akron 257

The Factorization of Large Composite Numbers on the MPP
Kathy J. McKurdy, Goodyear Aerospace Corporation
Dr. Marvin C. Wunderlich, Department of Defense 265

A Generic Applications Subroutine Library for the MPP
Michael L. Gough and
David W. Wildenhain, Science Applications Research 271

MPP Parallel FORTH 275
Dr. John E. Dorband, NASA GSFC

APPENDIXES

Appendix A: Index of Authors 285

Appendix B: Technical Summary 289

Appendix C: Symposium Attendees 295

ILLUSTRATIONS

Plate I: Stereo Images of Digital Terrain

Plate II: Stereo Analysis Algorithms for Generating Topological Maps I

Plate III: Stereo Analysis Algorithms for Generating Topological Maps II

Plate IV: Ray Tracing on the MPP

Plate V: Animated Computer Graphics Models of Space and Earth Sciences Data

Plate VI: Algorithmic Commonalities in the Parallel Environment I

Plate VII: Algorithmic Commonalities in the Parallel Environment II

Plate VIII: MPP Symposium Speakers

EARTH SCIENCES SESSION I

Session Chair: Dr. Robert J. Gurney
NASA Goddard Space Flight Center

N87-26532

An Implementation of a Barotropic Quasigeostrophic Model of Ocean Circulation on the MPP

C.E. Grosch and R. Fatoohi

Old Dominion University
Norfok, Virginia
and
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia

Abstract

We discuss here the implementation on the MPP of a barotropic quasigeostrophic model of ocean circulation. The mathematical model, including scalings and boundary conditions is discussed. The numerical scheme, which uses compact differencing is also discussed. The implementation of this model on the MPP is then presented. Finally, some performance results are given and compared to results obtained using the VPS-32 and one processor of a CRAY-2.

Introduction

A central problem in the physics of ocean circulation is that of understanding the interactions of the spectrum of scales present in these motions. At the long wavelength end of the spectrum are the planetary scale motions, typified by the thermohaline gyre circulation, and the wind driven basin circulation. These dynamical features are reasonably quasisteady and are the background and, presumably, the forcing functions for the synoptic scale dynamics.

The synoptic scale motions are, roughly, on the scale of the radius of deformation and range from the meanders in the western boundary currents (one to two hundred kilometers), eddies (fifty kilometers), and fronts (ten kilometers or less) to the small scale eddies (one kilometer or less) which appear to be in the dissipation range. These motions would seem to be generated by instabilities in the large scale gyre circulation which results in the transfer of either kinetic or potential energy from the gyre to the synoptic scale motion. From both a theoretical and practical point of view, an understanding of the synoptic scale dynamics appears to be of major importance.

The fundamental force balances of the ocean are hydrostatic in the local vertical and geostrophic in the horizontal. All interesting ocean dynamics are the results of the small deviations from these balances. Numerical simulations of these dynamics in terms of the "primitive" variables, velocity, pressure, and density, are difficult because, among other problems, of the required high accuracy of the solutions in order to separate the time varying dynamics from the quasistatic geostrophic and hydrostatic balances. However, a complete prediction of the dynamics, including the density field, requires the use of the primitive system (Bryan, 1975). The primitive variable general circulation models are the only ones which can give a complete prediction of the dynamics, including, especially, the density field. Currently available computer resources are not sufficient to allow fine horizontal and vertical resolution models to be used for routine parametric experiments with these models.

The standard approximation used to effect the separation is the quasigeostrophic model. In this approximation the dynamical variables are the horizontal velocity components and the vertical component of the vorticity. The density field appears only in the depth varying Brunt-Vaisala frequency. This approximation is extraordinarily rich in phenomena,

PRECEDING PAGE BLANK NOT FILMED

containing, for example, wind driven basin circulation dynamics, Rossby waves, barotropic and baroclinic instabilities, as well as eddies. Within the quasigeostrophic approximation there is much physics whose effects should be studied in detail; in some cases the physics has been explicitly built into the models while in others it is parameterized. Examples of the former are topography and irregular basin geometry, wind forcing and surface thermal forcing, both of which should be time dependent; the outstanding example of the parameterized physics is that of the dissipation mechanisms, or more generally, the effects of sub-grid scale turbulent motions.

The fundamental physics of the wind driven large scale ocean circulation were deduced by Sverdrup (1947), Stommel (1948) and Munk (1950). Analytic models such as these are limited by the complexity of the nonlinear dynamics. Further progress in understanding required the use of numerical models. Some examples of the early use of numerical models are provided by the work of Bryan (1963), Veronis (1966) and Holland (1967). More recently Bretherton and Darweit (1975), Bryan (1975), Haidvogel, Robinson, and Schulman (1980), Robinson and Haidvogel (1980), Haidvogel (1983), Holland, Harrison, and Semtner (1983) and Miller, Robinson, and Haidvogel (1983), among others, have developed numerical models of quasigeostrophic flow and used them to study ocean dynamics.

In their most general form these models require very large amounts of computation time on the fastest computers available. Despite the fact that the most powerful existing vector processors can perform at peak rates of hundred of MFLOPS, they are inadequate for many applications. In part this inadequacy is because it is generally rather difficult to fully use the vector capabilities of these supercomputers. In practice, average processing rates for many codes are in the range of 10 to 20 MFLOPS (see, for example, Dongarra, 1984). An alternative way of achieveing greater processing power is to use computers consisting of multiple (hundreds or thousands) processing elements; each processing element has only a modest processing power and storage. However, a complete multiple processing system can have a very large processing capability. If these multiprocessor computers can be used effectively, very large gains in overall processing power are possible.

In this paper we report on the implemenation of a quasigeostrophic potential vorticity model on the Massively Parallel Processor (MPP). In section 2 we give the scalings, basic assumptions, equations, and boundary conditions. The numerical scheme is outlined in section 3. Section 4 contains a discussion of the implementation of this model on the MPP. Finally, in section 5 the preliminary results of this research are given and some conclusions are drawn.

Equations

The scaling of the variables and the derivation of the quasigeostrophic equations follows the development of Pedlosky (1979). The x axis is positive eastward and the y axis is positive northward. The coriolis parameter is approximated by the beta plane approximation.

$$f = 2\Omega \sin\phi \approx f_0 + \beta_0 \tilde{y}, \qquad (2.1a)$$

with

$$f_0 = 2\Omega \sin\phi_0, \qquad (2.1b)$$

$$\beta_0 = (2\Omega/r_0)\cos\phi_0, \qquad (2.1c)$$

where Ω is the rotational frequency of the earth, r_0 is the radius of the earth, and $\tilde{y}$ is the dimensional distance northward of the latitude ϕ_0 (see Figure 1). A lateral length scale, L, and a vertical length scale, D, are used. It is assumed that D/L is much less than unity. The horizontal velocity components, u to the East and v to the North are scaled by U. The time is scaled by $U/\beta L$ where

$$\beta = \beta_0 L^2/U. \qquad (2.2)$$

Expanding the solution in powers of the Rossby number(U/foL), the equations of the quasigeostrophic model for the depth averaged horizontal velocity $\vec{U}$ =(u,v) and vertical component of the vorticity, ζ are:

$$u_x + v_y = 0$$

$$v_x - u_y = \zeta$$

$$\zeta_t + S(u\zeta_x + v\zeta_y) + \gamma\zeta + v =$$

$$\hat{k}\cdot(\nabla\times\vec{\tau}) + SRe^{-1}(\zeta_{xx} + \zeta_{yy})$$

$$S = \beta^{-1}, \quad \gamma \text{ is} \tag{2.5}$$

dimensionless bottom friction coefficient and $\vec{\tau}$ is the surface wind stress. The wind stress is dimensionless and its magnitude, τ_0, has been scaled to one by taking

$$U = \tau_0/\rho D\beta_0 L. \tag{2.6}$$

Finally the Reynolds number, Re, is defined by

$$Re = UL/A, \tag{2.7}$$

with A a horizontal eddy viscosity. In this derivation it was assumed that the density was constant, that the bottom was flat and that the wind stress, $\vec{\tau}$, was applied to a rigid lid on the surface.

If these equations are applied to the flow in a closed basin the boundary conditions are quite simple, $\vec{U}$=0 on the boundary. This implies that the vorticity on the boundary is equal to the normal derivative of the velocity on the boundary.

For application to flow in basins which are partly or completely open, inflow and outflow boundary conditions must be imposed. This is a subject of continuing research and will not be discussed further.

It should be noted that this formulation of the barotropic quasigestrophic equations is in terms of velocity and vorticity. This is in contrast to most other formulations in which a stream function is introduced to satisfy equation (2.3). Among the few to use the velocity-vorticity formulation in numerical calculations are Dennis, Ingram, and Cook (1979), Fasel (1980), and Gatski, Grosch, and Rose (1982). The velocity, vorticity formulation of the Navier-Stokes equations has certain advantages over more conventional formulations, particularly with respect to setting boundary conditions. For example, it is quite easy to incorporate the generation of vorticity at solid boundaries, and at outflow boundaries to use a vorticity flux condition (Halpern, 1985) to ensure that no vorticity is artificially reflected back into the computational domain.

Numerical Scheme

The numerical method used is based on compact differencing schemes. These schemes require the use of only the values of the dependent variables in and on the boundaries of a single computational cell. Compact difference methods have been discussed by Keller (1974). Malik, Chuang, and Hussaini (1982) used a fourth-order, compact, difference scheme to solve the compressible linear stability equations, and Gatski, Grosch, and Rose (1982) applied second-order compact schemes to the numerical solution of the incomprehensible, two-dimensional, time-dependent Navier-Stokes equations. This code was later extended to three dimensional time-dependent flows and used to study some complex flow fields (Gatski and Grosch, 1985a,b). The original ocean circulation code was developed as a serial algorithm by Spence and Grosch (1985).

In order to apply the compact differencing scheme we must write the vorticity equation as a first order system. Define

$$\phi = \zeta_x \tag{3.1a}$$

$$\psi = \zeta_y \tag{3.1b}$$

then (2.5) becomes

$$\zeta_t + S(u\zeta_x + v\zeta_y) + \gamma\zeta + v = \tag{3.2}$$

$$\hat{k}\cdot(\nabla\times\vec{\tau}) + SRe^{-1}(\phi_x + \psi_y).$$

The velocity and vorticity at time levels $n\Delta t$ and $(n\pm\frac{1}{2})\Delta t$ are located on the boundaries of and in a cell as shown in Figure 2.

The centered difference and average operators are defined on a cell by

$$\delta_x U^n \equiv (U^n_{i+1/2,j} - U^n_{i-1/2,j})/\Delta x, \quad (3.3)$$

$$\mu_x U^n \equiv (U^n_{i+1/2,j} + U^n_{i-1/2,j})/2. \quad (3.4)$$

With these definitions the, equations (2.3), (2.4) replaced by

$$\delta_x U^n + \delta_y V^n = 0, \quad (3.5)$$

$$\delta_x V^n - \delta_y U^n = \zeta^{n-1/2}, \quad (3.6)$$

$$\mu_x U^n - \mu_y U^n = 0, \quad (3.7)$$

$$\mu_x V^n - \mu_y V^n = 0 \quad (3.8)$$

The vorticity transport system, equations (3.1a), (3.1b), and (3.2) are replaced by

$$[\delta_t + S(\mu_x U^n)\delta_x \quad (3.9)$$

$$+ S(\mu_y V^n)\delta_y + \gamma]\,\zeta^n$$

$$= F^n - \mu_x \mu_y V^n + S\,Re^{-1}[\delta_x \phi^n + \delta_y \psi^n],$$

$$\delta_x \zeta^n = (\mu_x - \tfrac{1}{2}\Delta x\, q(\theta_x)\delta_x)\,\phi^n, \quad (3.10)$$

$$\delta_y \zeta^n = (\mu_y - \tfrac{1}{2}\Delta x\, q(\theta_y)\delta_y)\,\psi^n, \quad (3.11)$$

$$\mu_t \zeta^n = \mu_x \zeta^n = \mu_y \zeta^n, \quad (3.12)$$

and

$$q(\theta) \equiv \coth\theta - 1/\theta, \quad (3.13)$$

with

$$\theta_x = S\,U^n Re\,\Delta x/2, \quad (3.14)$$

$$\theta_y = S\,V^n Re\,\Delta y/2, \quad (3.15)$$

the cell Reynolds numbers. Note that $F^n_{i,j}$ is the cell averaged curl of the wind stress at time $n\Delta t$.

The boundary condition for the velocity equations (3.5) to (3.8) are that the normal component is zero on a solid boundary. This produces a nonzero tangential component on the boundary, which in turn gives the boundary condition for the vorticity equations (3.9) through (3.15); enough vorticity is produced at the wall so that the tangential velocity is brought to zero.

Implementation on the MPP

In order to map the data onto the MPP, it is convenient to introduce auxiliary or box variables to represent the velocity field. Referring to figure 2, we introduce the box variables P and Q at the corners of the cells. These are defined so that the average of two adjacent P's is equal to the U on the included side and similarly for the Q's and V. We then map this cell onto the array so that each corner of the cell "corresponds" to a processor and holds a P and a Q value. Again referring to Figure 2, if one considers the processor at the upper left hand corner of the cell the value of the vorticity at the center of the cell and on the left hand and bottom sides, as well as the corresponding values of ϕ and ψ, are stored in that processor. Thus, the array with 128 by 128 processors is mapped onto a 127 by 127 array of cells.

It is easy to see that, expressing U and V in terms of the P's and Q's, equations (3.7) and (3.8) are satisfied identically. One must then solve (3.5) and (3.6) with given and the boundary condition that the normal component of the velocity is zero on the boundary of the domain. This is done using a cell relaxation scheme in which all the values of P and Q on the corners of a cell are updated simultaneously, for details see Gatski, et al. (1982). It is obvious that the values of P and Q at the corner of any cell are updated four times in a sequential sweep across the domain. Thus, this method is a four "color" scheme. It is quite simple to update each of the "colors" in parallel and four updates are required for a complete update of all the points in the domain. The relaxation scheme is equivalent to an SOR method.

Once a solution for the P's and Q's has been found, the next step in the solution procedure is to calculate the boundary values of the vorticity. This is done by computing the vorticity required to be added at the boundary so that the tangential component of the velocity is reduced to zero. This is done using equation (2.4).

Once the values of the vorticity on the boundary are known, one can proceed to solve the vorticity equations (3.9) through (3.15). Using equation (3.12) the vorticity at time level n+1/2 can be eliminated from (3.9). This, together with the other equations, gives an implicit system for the vorticity at time . This system is solved using an ADI method which requires the solution of tridiagonal systems of equations in, alternately, all rows and all columns. This is done in parallel on the array using the cyclic elimination algorithm. Once the vorticity at time is known as well as at (n-1/2) , equation (3.12) can be used to compute the vorticity at time (n+1/2) . Finally, various norms, the total energy and so on are computed. The mathematical details of this algorithm are described in Gatski et al. (1982).

Results and Discussion

The algorithm described above has been coded in MPP Pascal and run on the MPP. The code is moderate length; between 1300 and 1400 lines. This can be compared with the serial FORTRAN version which is about 2000 lines long. The code for the four color relaxation is simple and highly structured. In contrast the code for the ADI vorticity solver is much more complex and less structured, it also uses about 20 temporary parallel matrices. There are procedures to calculate an approximation to the q function, equation (3.13) and to solve tridiagonal equations over the rows and columns. Despite this complexity, the time used by the ADI solver is only equal to ten or so iterations of the velocity solver. When the wind stress is varying in time or there are Rossby waves propagating in the basin the velocity solver requires a few hundred iterations per time step and, thus, uses most of the computational time. Therefore the performance of the parallel relaxation routing largely determines that of the entire code.

Processing rates, usually expressed in megaflops, are one measure of the performance of an algorithm on a computer. Comparison of the processing rates of the same algorithm on different computers can be misleading because of the differences in the basic cycle time of the machines and because the same algorithms may require different numbers and kinds of operations when optimized for different architectures. Neverhtless, cross comparisons are useful when a comparison is also made with the maximum theoretical processing rate of the computers. This kind of comparison shows how well, or poorly, a particular algorithm has been adapted to each of the architectures. Such a comparison is shown in Table 1.

The relaxation algorithm, in addition to being programmed for the MPP, has been programmed in CDC Vector FORTRAN for the VPS-32, and in FORTRAN for the CRAY-2. The programs were run on each of these machines and timings were obtained. These, together with the operation counts from the codes permitted the arithmetic processing rates to be determined. These together with the theoretical maximum rates are given in Table 1 for a 128 x 128 grid point problem.

The scalar rates shown in this table were obtained when standard FORTRAN codes embodying a serial version of the algorithm were run through a vectorizing compiler. None of the code was vectorzied. The vector performance on the VPS-32 and the CRAY-2 was obtained after the FORTRAN codes were rewritten, explicitly using the four color structure of the algorithm. In addition, the explicit Vector FORTRAN was used for the VPS-32 code and the CRAY compiler had to be told to ignore apparent vector dependencies.

This relaxation algorithm mapped well onto the MPP. This is because it can be set up as nearly all matrix operations. There are relatively few shift operations and these are all to nearest neighbors. There are no vector operations and only two scaler operations per iteration.

In contrast this algorithm does not vectorize as well. Because of the color structure, the longest vectors are one-half of the maximum dimension; in this case 64. This contrasts with N-1/2 value of 110 for the VPS-32. The structure of the algorithm also requires that the data be accessed with a stride of 2 and this requires the extensive use of COMPRESS and MERGE functions. This adds a considerable overhead. Increasing the problem size will improve the vector length problem but will not remove the overhead associated with the slow COMPRESS and MERGE functions.

On the CRAY-2 the stride of 2 is a source of major difficulty. All of the inner loops of the rewritten code vectorized, but the major problem was loading the data from the main memory to the local memory or the registers. Here the stride of 2 caused bank conflicts.

A perhaps more realistic way of measuring performance is to consider the time required to solve a given problem. Table 2 contains a listing of the measured time required to complete one iteration of the 128 x 128 problem on the MPP, the VPS-32, and the CRAY-2. The execution time for the CRAY is about 30% less than that of the MPP, while that of the VPS-32 is nearly 2.5 times greater than that of the MPP. We also note that we do more operations on the MPP than on the CRAY.

A crucial question in using the MPP is whether or not a problem will fit on the array or is oversize. Each of the major procedures in this code requires 20 to 25 arrays of floating point numbers. Many of these are temporary arrrays, but these together with other storage require that we move various arrays in the out of the staging memory even when we have a 128 x 128 problem. Therefore, we must partition these oversize problems and use the staging memory as a backup. We have modified the algorithms so as to be able to handle oversize algorithms, the details will appear elsewhere. A total of seven data swaps between the stager and the array are required for each sheet of data. The swapping adds 11.5 msec to the time for each iteration; thus one iteration on a sheet requires 25.1 msec. Some results for oversize problems, both measured and extrapolated, are given in table 2.

In summary we have found that it is possible to adapt a barotropic quasigeostrophic potential vorticity code to the MPP. We have been able to use the MPP's considerable processing power to solve the computationally intensive flow problems of ocean circulation. For problems which fit on the array, one iteration requires about 14 msec; this is about 30% more than that required on one processor a CRAY-2, and only about 40% of the time required for the same calcualtion on a VPS-32. For oversize problems, 2, 3, ..., 10 times the array size there is an overhead for transferring the data to and from the staging memory. This reduces the efficiency of the MPP, but not drastically. It is still competitive with the other supercomputers.

The problems which we have encountered in using the MPP have not been major. With regards to software, MPP Pascal as currently implemented, does not permit operations on vectors. This is awkward when dealing with boundary conditions. The major hardware problem is that the PE memory is only marginally large enough for our applications. We really need 4K to 8K bits per PE for the barotropic code and 32K bits per PE for a baroclinic model. Finally, input/output is somewhat clumsy.

Despite these problems, we believe that the MPP is a useful tool for running ocean circulation models and we will continue using it to study the effects of temporal and spatial variation in the wind field.

REFERENCES

Bretherton, F.P., and M. Karweit, "Mid-Ocean Mesoscale Modeling," In: *Numerical Models of Ocean Circulation*, National Academy of Sciences, 94, 1975.

Bryan, K., "Three-Dimensional Numerical Models of the Ocean Circulation," In: *Numerical Models of Ocean Circulation,* National Academy of Sciences, 94, 1975.

Dennis, S.C.R., D.B. Ingram, and R.N. Cook, "Finite Difference Methods for Calculating Steady Incompressible Flows in Three Dimensions," *Journal of Computational Physics,* **33**, 325, 1979.

Dongarra, J.J., "Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment," Argonne National Laboratory Technical Memorandum 23, 1984.

Fasel, H.F., "Numerical Solution of the Complete Navier-Stokes Equations for the Simulation of Unsteady Flows," Lecture Notes in Mathematics, Number 771, Springer-Verlag, New York/Berlin, 1980.

Gatski, T.B, and C.E. Grosch, "A Numerical Study of the Two- and Three-Dimensional Unsteady Navier-Stokes Equations in Velocity- Vorticity Variables Using Compact Schemes," *Ninth International Conference on Numerical Methods Fluid Dynamics*, Springer-Verlag, 235, 1985a.

Gatski, T.B., and C.E. Grosch, "Embedded Cavity Drag in Steady Laminar Flow, " *AIAA Journal*, **23**, 1028, 1985b.

Gatski, T.B., C.E. Grosch, and M.E. Rose, "A Numerical Study of the Two-Dimensional Navier-Stokes Equations in Vorticity- Velocity Variables," *Journal of Computational Physics*, **48**, 1, 1982.

Haidvogel, D.B., A.R. Robinson, and E.E. Schulman, "The Accuracy, Efficiency, and Stability of Three Numerical Models with Applications to Open Ocean Problems," *Journal of Computational Physics*, **34**, 1, 1980.

Haidvogel, D.B., "Periodic and Regional Models," In: *Eddies in Marine Science*, Ed. A.R. Robinson, Springer-Verlag, 404, 1983.

Halpern, L., "Artificial Boundary Conditions for the Linear Advection Diffusion Equation," CMAP-Ecole Polytechnique, Paris Report #118, 1985.

Holland, W.R., D.E. Harrison, and A.J. Semtner, Jr., "Eddy-Resolving Numerical Models of Large-Scale Ocean Circulation," In: *Eddies in Marine Science*, Ed. A.R. Robinsons, 379, 1983.

Keller, H.B., "Accurate Difference Methods for Nonlinear Two- Point Boundary Value Problems, " *SIAM Journal on Numerical Analysis*, **11**, 305, 1974.

Malik, M.R., S. Chuang, and M.Y. Hussaini, "Accurate Numerical Solution of Compressible Linear Stability Equations," *Journal of Applied Mathematical Physics*, (ZAMP), **33**, 189, 1982.

Miller, R.N., A.R. Robinson, and D.B. Haidvogel,, "A Baroclinic Quasigeostrophic Open Ocean Model," *Journal of Computational Physics*, **50**, 38, 1983.

Munk, W.H., "On the Wind-Driven Ocean Circulation, " *Journal of Meteorology*, **7**, 79, 1950.

Pedlosky, J., *Geophysical Fluid Dynamics*, Springer-Verlag, Berlin, 1979.

Robinson, A.R., and D.B. Haidvogel, "Dynamic Forecast Experiments with a Barotropic Open Ocean Model," *J. Phys. Ocean*, **10**, 1909, 1980.

Robinson, A.R., "Overview and Summary of Eddy Science," In: *Eddies in Marine Science*, Ed. A.R. Robinson, Springer-Verlag,3, 1983.

Stommel, H., "The Westward Intensification of Wind-Driven Ocean Currents," *Trans. Amer. Geo. Union*, **29**, 202, 1948.

Sverdrup, H.V., "Wind Driven Currents in a Baroclinic Ocean; With Application to the Equatorial Currents of the Eastern Pacific," *Proceedings of the National Academy of Sciences*, **33**, 318, 1947.

	MPP	*VPS-32*	*Cray-2* (*)
Theoretical Maximum Rate	210	200	488
Measured Vector or Array Rate	175	48	102
Measured Scalar Rate	--	7	30

(* Using one processor)

Table 1. Theoretical and measured processing rates, in Megaflops, using the relaxation algorithm on a 128 x 128 grid.

	MPP	*VPS-32*	*Cray-2*
Problem Size			
128 x 128	13.6	33.4	10.4
128 x 255	50.2	47.9	20.7
128 x 509*	100.4	73.6	41.4
128 x 1017*	200.8	124.0	82.6

(* Computed by extrapolation)

Table 2. Execution time, in milliseconds, for one iteration. One processor the CRAY-2 was used.

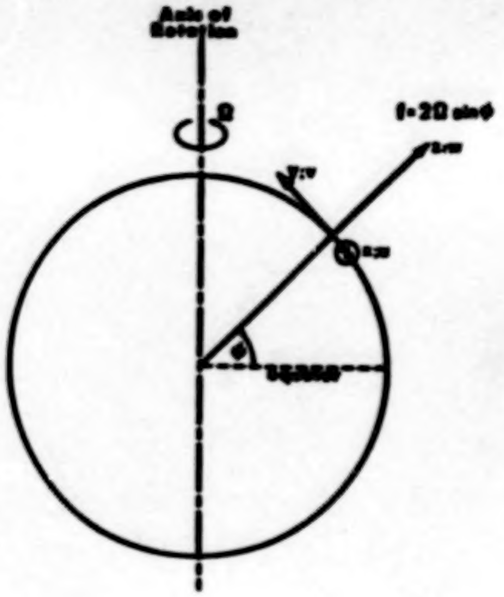

Axis of Rotation
Ω
f=2Ω sinϕ
z;w
y;v
x;u
ϕ
Equator

Figure 1. Coordinate System

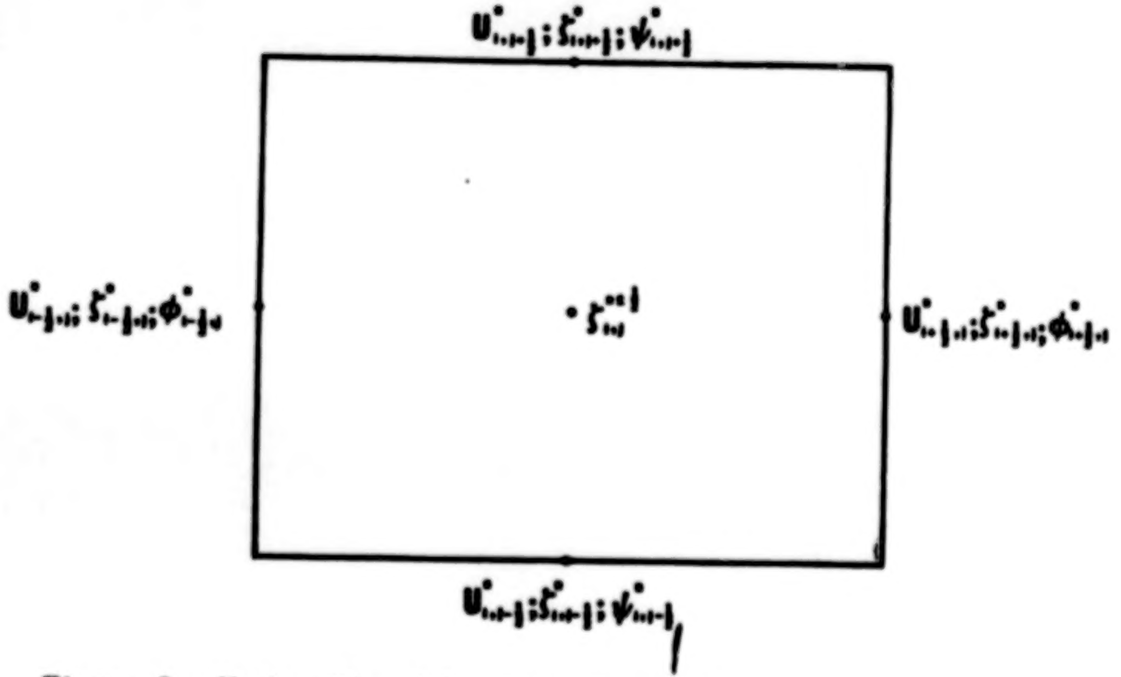

U*i,j+½ ; ζ*i,j+½ ; ψ*i,j+½
U*i-½,j ; ζ*i-½,j ; ϕ*i-½,j
ζn+1 i,j
U*i+½,j ; ζ*i+½,j ; ϕ*i+½,j
U*i,j-½ ; ζ*i,j-½ ; ψ*i,j-½

Figure 2. Mesh cell and location of variables.

N87-26533

COMPARISON OF THE MPP WITH OTHER SUPERCOMPUTERS FOR LANDSAT DATA PROCESSING

by
Martin Ozga
United States Department of Agriculture
National Agricultural Statistics Service

ABSTRACT

The MPP is compared to the CRAY X-MP and the CYBER-205 for Landsat data processing. The maximum likelihood classification algorithm is the basis for comparison since this algorithm is simple to implement and vectorizes very well. The algorithm was implemented on all three machines and tested by classifying the same full scene of Landsat MSS data. Timings are compared as well as features of the machines and available software.

INTRODUCTION

The National Agricultural Statistics Service (NASS) of the United States Department of Agriculture has for several years been using Landsat MSS (multispectral scanner) data to aid in crop acreage estimation. The estimates are provided for several states, mostly in the Midwest. For each state, as much Landsat data as possible is obtained, the ideal being coverage for the entire state. Therefore, large amounts of data must be processed. This large-scale processing has long been seen as a useful application of supercomputers, first the ILLIAC-IV and more recently the CRAY X-MP (Ref. 1). One of the key programs in the analysis is the classification or categorization of the data into classes representing crop types. This classification is done on all scenes for all states.

LANDSAT MSS DATA

Each Landsat MSS scene consists of about 10.5 million pixels. Each pixel covers a square on the earth's surface of about 57 meters on a side. Each pixel consists of the reflectance intensity in 4 spectral bands or channels. The intensity is a value between 0 and 255 (more commonly between 0 and 127) and thus occupies 1 byte of data. Thus each pixel occupies 32 bits of data, 1 word on many machines or a half word on many supercomputers.

Often, to improve classification accuracy, multitemporal data are used. A multitemporal data set is made up of two overlaid Landsat data sets from the same area on the earth's surface. The improvement comes in that the two data sets are taken at different times and, if the dates are chosen properly, may emphasize differences in reflectance of various crops. However, since the multitemporal data sets contain eight channels of data, they do take longer to process.

For the USDA-NASS 1986 analysis, approximately 80 scenes of Landsat data will be used of which approximately 25 will be multitemporal. For the purposes of the test described in this paper, only unitemporal data sets were used.

THE CLASSIFICATION ALGORITHM

The maximum likelihood classification algorithm consists of applying a discriminant function for each class to each pixel and assigning the pixel to the class for which the discriminant function yields the highest value. Since each pixel is processed independently, the algorithm is ideal for vector machines and indeed performs well on all machines tested. The discriminant functions is:

$$G(X, I) = B(I) - .5\,(X - M(I))^{T}\,V(I)\,(X - M(I))$$

where X is the pixel
I is the class
M(I) is the vector of mean values for class I
V(I) is the inverted variance-covariance matrix for class I
B(I) = .5 log (determinant (V (I))

PRECEDING PAGE BLANK NOT FILMED

There are three parts to the algorithm. First, the data must be unpacked and converted to floating point form. That is, each byte (channel intensity value) must be converted to the equivalent floating point value. Second, the discriminant function must be applied for each class to each pixel and the class for which the discriminant function yields the highest value saved for each pixel. Third, the data are repacked to one byte per pixel for economy in storage for further processing. This means that the number of classes must be less than or equal to 255; in practice at USDA-NASS the number of classes is always well under 255. Obviously the second step is by far the most time consuming.

MPP IMPLEMENTATION

The MPP classification algorithm was implemented on the MPP using Parallel Pascal (Ref. 2, 3, 4). The entire algorithm was implemented without requiring use of assembly language so Parallel Pascal was certainly adequate to the task, although, perhaps coding the key loop of discriminant functions evaluation in assembly language may have speeded up the algorithm somewhat.

The input-output was done using supplied fast video routines callable from Parallel Pascal. The data was mapped so that all four channels of each pixel were transferred to the same PE. Of the 1024 bits available in each PE, 512 were used for pixel storage allowing 16 pixels to be stored in each PE at any one time, or 262144 pixels in the entire PE memory.

Thus, 42 I/O operations were necessary to bring in all the data. When the program was written and debugged, facilities were not available for overlapping processing and I/O. The class associated with each pixel was stored over a portion of the original pixel values since once the pixel was converted to floating point, the original values were not needed any longer. An additional 128 bits were needed for the four 32-bit floating point channel values. Additional bits were used for various temporary values. Some space was left available for a possible future expansion to multitemporal, eight-channel data.

It is evident that when processing 10.5 million pixels at 16384 at a time, all PE's are generally doing productive work. The lack of hardware floating point did slow this algorithm. Some tests on the CRAY X-MP with integer processing revealed that integer processing led to severe deterioration of classification results unless certain intermediate results were stored is very large integers, being due to the significant digits which must be maintained using the inverted variance - covariance matrix.

The repacking of data, a rather tedious tasks on the other machines, was easily realized by proper storage of the data on the MPP and correct maps for sending the data through the staging buffer to the output categorized file.

CRAY X-MP IMPLEMENTATION

The CRAY X-MP/48 at NASA-Ames was used for the test. The program, which is in production use by USDA-NASS, is coded in Cray FORTRAN (CFT) (Ref. 5) with key routines coded in CRAY assemble Language (CAL) (Ref. 6). The use of assembly language is mainly for historical reasons since at the time this program was originally written, the CFT compiler was less well developed than it is today. If this program were re-written, much more of it would be coded in CFT.

The CRAY X-MP vectorization uses as operands vector registers each of 64 64-bit words. The vectorization is achieved largely through pipelining data. There is some advantage to having longer vectors, particularly when coding in CFT, since overhead for vector set-up is decreased. A vector size of 16384 was used for the CRAY implementation.

CYBER-205 IMPLEMENTATION

The Control Data CYBER-205 at NASA-

Ames was used for the test. The program was coded in FORTRAN (Ref. 7), but makes extensive use of special extensions and subroutines. This was necessary since at the time the program was written and debugged, the CYBER-205 FORTRAN compiler did a rather poor job of vectorization, a condition which is being gradually improved.

Vector operations on the CYBER-205 operate directly out of memory and may be of indeterminate length, limited by the size of memory. Vectorization on the CYBER-205 is achieved largely by pipelining data. The CYBER-205 implements virtual memory with paging so there is a potential loss of efficiency if a vector crosses a page boundary or if two operands to a vector operator are in different pages in that a page fault may cause a break in the vector operation. With these constraints in mind, the CYBER-205 operates best on long vectors. Also, internally, vector operands are pointed to by descriptors which may be used explicity by the FORTRAN programmer desiring maximum efficiency. Finally, CYBER-205 FORTRAN provides many subroutine calls to perform vector operations which are not directly expressed in FORTRAN, that is for most of the capabilities of the machine outside of ordinary arithmetic operations. In effect, when all these features are used, one has a parallel FORTRAN, somewhat analogous to Parallel PASCAL on the MPP.

The classification algorithm was coded on the CYBER-205 using descriptors, several of the special subroutines, and with consideration for paging. The vector length was 16384. Such a coding is in the parallel form of FORTRAN and it provides for an efficient use of the machine.

CPU TIMES

The test performed involved classifying a full scene of data (10.5 million four channel pixels) into 51 categories. The same results, as determined by a count of the number of pixels per category, were obtained on all machines except for a difference of two pixels on the CYBER-205, an insignificant difference.

The following CPU times were observed:

MPP: 90 seconds (approximate)
CRAY X-MP: 157 seconds
CYBER-205: 58 seconds.

It is interesting to note that the MPP lies between the CRAY X-MP and the CYBER-205 in timings. Improvements in the code generated by Parallel Pascal or the use of assembly language would no doubt bring the timing down closer to that of the CYBER-205; similar improvements on the CYBER-205 are unlikely since the code already takes advantage of special features of the CYBER-205. One would suppose that implementation of hardware floating point operations on the MPP would cause it to run faster than the CYBER-205 on this problem.

Both the MPP and the CYBER-205 are known to operate best on long vectors whereas the CRAY X-MP has a shorter vector but faster scalar speed. The classification algorithm certainly shows the advantages of the long vectors when applied to an algorithm for which they can be profitably used.

The CPU times for the CRAY X-MP and CYBER-205 were obtained directly from the listing received from batch execution of the program. Such a listing was not available for the MPP. Therefore, calls to timing routines were placed around the main classification loop. These timings totaled 88.76 seconds. The CPU times spent in the remainder of the program would be quite small.

ELASPED TIMES

Elapsed times on the CRAY X-MP and CYBER-205 are not important since both machines are operated in a multiprogramming environment so the elapsed time would be very dependent on the mix of jobs in the system.

However, only one job is present at any one time on the MPP so that the MPP user would expect to be charged for the entire elapsed time. The total job time on the MPP was 360 seconds. This time fluctuated somewhat, presumably depending on usage of the front-end VAX. The test was run at a time of relatively low VAX use, on a Friday evening. Since I/O-CPU overlap was not available, the time spent in reading the input file was measured to try to get an approximation of savings to be obtained if overlap were present. The input file was used since it is largest and only the input file was used since only one I/O operation may proceed at any one time. Approximately 150 seconds was spent reading the input file indicating that the program was I/O bound. Thus, if CPU and input I/O could be overlapped by reading smaller pieces of the input file into buffers in PE memory, one would expect the total job in the decrease to about 270 seconds. The time not accounted for in the above calculations, 120 seconds, was spent in writing the output file and also in reading the statistics file and preassembly in some overhead in communicating with the VAX. Thus, one would, in a production mode, expect to pay for 270 seconds of MPP time to do a 90 second job. Of course, on the CRAY X-MP and CYBER-205 one is typically changed for I/O and often memory usage. The wide variance in applying these changes from site to site makes comparison difficult.

EASE OF USE

Ease of use is necessarily subjective term, based on one's opinions and experiences with various system. Nevertheless, some comments can be made. I found the MPP generally easy to use. Parallel Pascal provides a good interface to the machine. The biggest problem in Pascal and other languages is that one must often use special routines for I/O. Once I learned about the special routines, I found them generally convenient to use. The problem of using up the limited memory of the MPP with the stack space required for complicated expression can be severe; perhaps an option should be added to Parallel Pascal to tell the user how much stack space is used by any particular statement. Otherwise, at least for this particular problem, the size of the PE memory was not really a constraint.

Landsat processing is characterized by moving large amounts of data. While the facilities provided on the MPP were adequate for the test, they would not be adequate for production use. For the test, the Landsat data tape was copied on the VAX to a removable disk pack and this disk pack was then made available for the test. What is needed for production are facilities typically available at supercomputer sites wherein data are staged between tapes and dedicated, high-capacity, high-speed disks.

In general, I found both the software and hardware to be reliable. It was not necessary or even advantageous to use the MPP simulator since the MPP itself was generally available. Debugging facilities were perhaps a bit crude but adequate.

CONCLUSIONS

The MPP has promise for Landsat data processing. The most urgent need is for improved data handling. However, since the speed in this test was slower than on the CYBER-205, and since both are characterized by requiring long vectors, perhaps a more complex PE implementing hardware floating point is in order. It would be interesting to do a comparison on some algorithm requiring only or mostly integer computations. Unfortunately, none of the major programs currently used by USDA-NASS for Landsat processing fits this requirement so no such test was performed.

ACKNOWLEDGEMENT

The author would like to thank James Tilton of the NASA Goddard Space Flight Center for his generous help in answering my many questions and in locating appropriate subroutines, particularly for I/O.

REFERENCES

1. Ozga, Martin, "Experience with the Use of Supercomputers to Process Landsat Data," Symposium Proceedings,

Machine Processing of Remotely Sensed Data, Laboratory for Application of Remote Sensing, Purdue University, West Lafayette, Indiana, 1984.

2. Parallel Pascal Language Reference Manual, NASA Goddard Space Flight Center, 1986.
3. Parallel Pascal User's Guide, NASA Goddard Space Flight Center, 1986.
4. MPP Pascal Callable Procedure Library, NASA Goddard Space Flight Center, 1986.
5. FORTRAN (CFT) Reference Manual, SR-0009, Cray Research Inc., Mendota Heights, Minnesota, 1984.
6. CAL Assembler Version I Reference Manual, SR-0000, Cray Research , Inc., Mendota Heights, Minnesota, 1983.
7. FORTRAN 200 Version I, 60480200, Control Data Corporation, Sunnydale, California, 1984.

PROGRAMMING A HILLSLOPE WATER MOVEMENT MODEL ON THE MPP

J. E. Devaney, A. R. Irving
Science Applications Research
Lanham, Maryland

P. J. Camillo, R. J. Gurney
NASA Goddard Space Flight Center
Greenbelt, Maryland

ABSTRACT

We have developed a physically based numerical model of heat and moisture flow within a hillslope on a parallel architecture computer, as a precursor to a model of a complete catchment. Moisture flow within a catchment includes evaporation, overland flow, flow in unsaturated soil, and flow in saturated soil. Because of the empirical evidence that moisture flow in unsaturated soil is mainly in the vertical direction, flow in the unsaturated zone can be modelled as a series of one-dimensional columns. This initial version of the hillslope model includes evaporation and a single column of one-dimensional unsaturated zone flow. This case has already been solved on an IBM 3081 computer and is now being applied to the MPP architecture so as to make the extension to the one-dimensional case easier and to check the problems and benefits of using a parallel-architecture machine.

Keywords: Hydrology, Hillslopes, Parallel Processing, Unsaturated Flow, Evaporation.

INTRODUCTION

One important part of the global hydrological system is a catchment, which separates rainfall into three parts: evaporation, overland flow, which goes directly to a stream, and infiltration, which flows at a much slower rate vertically through the soil to a saturated zone, where the water then flows horizontally and merges into the stream or is stored as groundwater. This involves four components: evaporation, overland flow, flow in unsaturated soil and flow in saturated soil. If we are to understand the way in which spatial variations of hydrological parameters affect the water balance of a catchment, including evaporation, runoff, erosion and transport of minerals, we have to model the flow over and within a hillside explicitly.

Several workers have written hillslope models (Ref. 6). All of these have had limitations mainly because they could not be executed in a reasonable amount of time. This computer limitation is much reduced if a parallel processor is used, as it is possible to write the model in such a way that it can be solved in parallel at many points at one time.

The first stage of the work is to verify that such a model may be efficiently executed on a parallel processor. To this end we have coded and tested a model which already exists (Ref. 2) on a serial computer, an IBM 3081, and which contains two of the four previously identified components, namely evaporation and unsaturated flow. We have transferred an existing model to the MPP so that we may verify the accuracy of the parallel computations. We also plan to estimate the computational efficiency of the full catchment model in a parallel machine over the equivalent model on a serial machine, if such a serial model were to be successfully created.

In this paper we discuss the physics of the one dimensional model, the method of solution, and its adaptability to the parallel architecture.

Model Description

The complete soil model is described in Ref. 2. The soil moisture profile in the unsaturated zone is the solution of the continuity equation

$$\frac{\partial \theta}{\partial t} = - \frac{\partial q_m}{\partial z} \tag{1}$$

where $\theta(z,t)$ is the volumetric soil moisture (water volume/soil volume) at depth z at time t, and q_m is the vertical soil moisture flux, modelled by (Ref. 8)

$$q_m = K - D_\theta \left(\frac{\partial \theta}{\partial z}\right) \tag{2}$$

K is the hydraulic conductivity, and D_θ is the moisture diffusion coefficient, which depends on soil moisture θ, the soil matric potential ψ and other physical constants. K and ψ are estimated with the parameterization (Ref. 3)

$$K(\theta) = K_s(\theta/\theta_s)^{2b+3} \tag{3a}$$

$$\psi(\theta) = \psi_s(\theta/\theta_s)^{-b} \tag{3b}$$

in which θ_s, K_s, and ψ_s are moisture content, conductivity, and potential at saturation. The value of b depends on soil texture.

The temperature profile in the soil may be modelled with Fourier's equations and is one option in the serial version. However, we have chosen to implement the computationally simpler yet physically adequate force-restore equations (Ref. 7). The surface and deep soil temperatures T_s and $\bar{T}$ are modelled by

$$\frac{\partial T_s}{\partial t} = \frac{2G}{a} - \frac{2\pi}{\tau}(T_s - \bar{T}) \tag{4a}$$

$$\frac{\partial \bar{T}}{\partial t} = \frac{G}{a\sqrt{365\pi}} \tag{4b}$$

$$a = \sqrt{\lambda c \tau / \pi} \tag{4c}$$

where G is the soil surface heat flux, λ and c are respectively the soil thermal conductivity and heat capacity, and τ is the length of the day. $\bar{T}$ is the temperature at the depth where fluctuations are seasonal rather than diurnal. For most soils this depth is about 2 meters. The conductivity and heat capacity are modelled as functions of soil moisture and soil type with the model of Ref. 4.

To solve these equations for θ, T_s, and $\bar{T}$, boundary conditions must be supplied for moisture and temperature both at the air/soil interface and in the bottom layer of the profile. In principle, either the fluxes q_θ and G or the variables θ and T could be specified. In the model, surface heat and moisture fluxes are computed to model the effects of the environment (i.e., rainfall, evapotranspiration, radiation, etc.) on the profile evolution. At the bottom of the moisture profile a choice of flux or moisture boundary condition is used. One can specify constant moisture, a downward moisture flux equal to the hydraulic constant of the bottom layer, or any constant value.

The energy balance equation provides the surface fluxes:

$$G = R + LE + H \tag{5}$$

All fluxes are positive downward. G is the heat absorbed by the soil, R is

PRECEDING PAGE BLANK NOT FILMED

the net radiation flux, LE is the evapotranspiration energy flux, and H is the sensible heat. After finding the solution, the surface moisture flux q_θ is set equal to E and G is used in the force-restore equations.

The net radiation R is divided into average short- and long-wavelength components:

$$R = R_{short} + R_{long} \qquad (6)$$

Either or both components may be either estimated or measured. All four options are allowable within the computer program, standard models such as the Brunt model for long-wave radiation being available as options to estimate either or both components.

A standard model for the latent heat flux under neutral atmospheric stability is (Ref. 5)

$$LE = - \frac{\rho c_p k^2 \overline{U}_a}{\gamma \ln^2(z/z_o)}(e_s - e_a) \qquad (7)$$

where ρ is the density of air (1.15×10^{-3} g cm^{-3}), c_p the air specific heat, k the von Karman constant (0.4), z_o the surface roughness parameter, $\overline{U}_a$ the wind velocity (centimeters per second) at height z averaged over a suitable time period (~ 1 hour), e_a the vapor pressure at height z, e_s the vapor pressure at the soil surface, and γ the psychrometric constant (0.61808 mbar K^{-1}).

This may be expressed as

$$LE = -C_1 \overline{U}_a (e_s - e_a) \qquad (8a)$$

$$\text{where } C_1 = \frac{\rho c_p k^2}{\gamma \ln^2(z/z_o)} \qquad (8b)$$

Input to the program includes the constant C_1 and both $\overline{U}_a$ and e_a as functions of time.

The sensible heat flux H in the continuity equation (21) is calculated by

$$H = -\gamma C_1 \overline{U}_a (T_s - T_a) \qquad (9)$$

The terms of heat balance equations are functions of the known surface temperature T_s and the meteorological variables e_a, $\overline{U}_a$, and T_a.

METHOD OF SOLUTION

The continuity equations (1 and 4) are solved by expressing the spatial derivatives of the moisture fluxes as finite differences, and then using a fourth order predictor-corrector for the time integration.

The soil is divided into N layers of varying thicknesses Δz_i, where N and Δz_i are input variables. At a particular time the moisture fluxes at the N-1 interior boundaries are calculated by evaluating Eq. 2. The surface and bottom fluxes are computed by evaluating the boundary condition equations at the top and bottom boundaries. This gives the flux at the N+1 layer boundaries, and the derivative with respect to depth is approximated for the ith layer by

$$\left(\frac{\partial q_m}{\partial z}\right) \equiv \frac{q_{m_{i+1}} - q_{m_i}}{\Delta z_i} \qquad (10)$$

The continuity equations (1 and 4) are of the form

$$\frac{\partial \vec{y}}{\partial t} = \vec{f}\,(t, \vec{y}) \qquad (11)$$

where the state vector $\vec{y}$ has N+2 elements, soil moisture θ_i in each of N layers, T_s and $\overline{T}$. The first N elements of $\vec{f}$ are Eq. 10, and f_{n+1}

and f_{n+2} are the right hand sides of the force restore equations, 4a and 4b.

This is readily solved by an Adams-Bashforth predictor-corrector algorithm (Ref. 1,9). We first introduce the backward difference operator ∇ such that

$$\nabla\vec{f}(t) = \vec{f}(t) - \vec{f}(t - \nabla t)$$

Higher order backward differences are evaluated by successive applications of the operator;

$$\nabla^2\vec{f}(t) = \nabla(\nabla\vec{f}(t)) = \vec{f}(t) - 2\vec{f}(t-\Delta t) - \vec{f}(t-2\Delta t)$$

$$\nabla^3\vec{f}(t) = \vec{f}(t) - 3\vec{f}(t-\Delta t) + 3\vec{f}(t-2\Delta t) - \vec{f}(t-3\Delta t)$$

$$\nabla^4\vec{f}(t) = \vec{f}(t) - 4\vec{f}(t-\Delta t) + 6\vec{f}(t-2\Delta t) - 4\vec{f}(t-3\Delta t) + \vec{f}(t-4\Delta t)$$

Using only the state vector $\vec{y}$ and the physical model $\vec{f}$, at time t, an estimate (called the predictor) at time t+Δt is

$$\vec{y}^{(p)}(t+\Delta t) = \vec{y}(t) + \Delta t\left[1 + \frac{1}{2}\nabla + \frac{5}{12}\nabla^2 + \frac{3}{8}\nabla^3 + \frac{251}{720}\nabla^4\right]\vec{f}(t)$$

Then, using $\vec{y}^{(p)}(t+\Delta t)$ to evaluate the model $\vec{f}(t+\Delta t)$ at the t+Δt, one may compute another estimate of the state vector called the corrector;

$$\vec{y}^{(c)}(t+\Delta t) = \vec{y}(t) + \frac{251}{720}\Delta t\,\vec{f}(t+\Delta t) + \frac{\Delta t}{720}\left[469 + 109\nabla + 49\nabla^2 + 19\nabla^3\right]\vec{f}(t) \tag{13}$$

The physical model equations are evaluated only once each time step to calculate $\vec{f}(t+\Delta t)$. The difference between $\vec{y}^{(p)}$ and $\vec{y}^{(c)}$ is a reliable estimate of the error, and the software determines if each element of this difference lies within a user specified window. If all differences are smaller than this window, the integration step size (Δt) is doubled, leading to increased computational efficiency. If any difference is too large, the step size is halved. This halving and doubling requires no re-evaluation of the model equations. The values of the four backward differences for the new integrator time (whether for halving, doubling, integration) are calculated as linear combinations of the four back values for the old integrator time. The fact that the same calculations are done on many pieces of data at the same time as well as the boolean nature of the error window checks make the Hillslope Model an ideal application for a Single Instruction Multiple Data (SIMD) Massively Parallel Processor (MPP).

Mapping the Hillslope Model to the MPP Architecture

The initial mapping the Hillslope model to the MPP involved three phases:

1. Determination of scalar and parallel components of the calculations
2. Data initialization on the MPP
3. Data output to the VAX

Both calculations within the layers

of the hillslope and at the boundaries are required at each time step of the model integrations. The calculations within the hillslope are exactly the same at each layer, so they were set up so that they could be done in unison by means of parallel arrays in the MPP Array Unit. These parallel values consisted of the soil moisture, temperature, depth, layer thicknesses, fluxes, derivatives, backward differences, as well as predicted and corrected values. Because the model was set up with a view to extension to a two dimensional model, each row in the Array Unit was to represent one vertical column in the hillslope, with the components of the row representing the layers in it. This initial version of the model was set up with 14 layers in the soil column to match the serial calculations. Thus in this one-dimensional rendition, only part of the first row of the Array Unit was used for each parallel array of data. In addition, since the temperature and moisture values constitute the state vector $(\vec{y})$ and thus require the same type of computations, they were placed together in the first row of the array unit. While this is initially wasteful in terms of the computational power of the MPP, it allows an easy extension to a two dimensional model which will use the other rows and hence the full capabilities of the MPP.

The boundary conditions involve only single values at the top and bottom of the soil column modified with scalar input data, and so they may be more efficiently done serially with the Main Control Unit (MCU). Thus, implementing the model on the MPP involved scalar and parallel components as well as communication between the scalar and parallel components for boundary condition values. The MPP architecture is very efficient at passing scalar values back and forth between the MCU and the Array Unit as it has a register designed for this purpose. Two special purpose assembly routines were written to take advantage of this. One took a user specified row and column in a parallel array and placed a given scalar value there. Another took a user specified row and column in a parallel array and retrieved a value into a scalar in the MCU. These were used in the boundary condition calculations.

Data intialization on the MPP therefore involved initialization of parallel arrays as well as scalar data. These arrays were initialized in FORTRAN arrays and scalars on the VAX and transferred over to the MPP scalars and arrays via the DR780 and DR11b buffers. The capability of the stager to permute the data bits between the VAX and the Array Unit was used to change the format of the floating point data from the VAX format to the MPP format while the data was being transferred between the VAX and the MPP. Because of this, parallel data transmission between the VAX and the MPP appeared transparent. Explicit bit swapping of scalar integer data between the VAX and MCU to accomodate the two separate integer formats was still necessary, however, as these values were transmitted across the DR11b which has no data permuting capability.

Special purpose routines were written to enable data output to the VAX of the information in the parallel arrays. These routines passed parallel arrays into predefined VAX FORTRAN arrays. MPP Pascal callable VAX FORTRAN routines were written which could write out the data in these arrays for user examination of intermediate and final results. In addition, some high level I/O equivalent to Pascal 'writeln' was written to speed the debugging process by bypassing the CAD debugger entirely.

The next step in mapping the Hillslope model to the MPP architecture involved implementing the physical model equations in MPP Pascal.

The predicted and corrected state vector values (soil moisture and temperature) involve a calculation of the type (see Eqs. 12 and 13)

$$\vec{y}_{n+1} = \vec{y}_{n} + \Delta t \sum_{i} W_{i} * tp_{i}$$

where

W_i are the backward differences for each layer

tp_i are the constant scalar coefficients in the predictor equation

By assigning the backward differences to individual parallel arrays so that order ith backward differences for each layer are stored in the first row of the ith parallel array, the above calculation can be solved for each layer with a series of parallel operations. $W_i * (tp_i * \Delta t)$ involves only multiplication of a parallel array by a scalar as the coefficients of the predictor equation would be the same for each layer with the same order backward differences. The remaining sums can be done for each layer in parallel. Once the predicted and corrected values of each layer are available, their differences can be simultaneously calculated. Comparisons with user input error windows can be done all at once with simple boolean tests on the parallel arrays. Recalculation of the backward differences for each layer can be done in unison with the backward difference arrays.

There are additional calculations needed at each layer and each time step in order to include the dependencies of the heat and moisture fluxes on the moisture and temperature profiles through the column. These involve the matric potential, the hydraulic conductivity, the moisture fluxes as well as the derivatives of the moisture fluxes and temperatures. Special calculations must be done at the boundaries in order to include the effects of the air/soil interface at the top as well as the effects of saturated soil at the bottom of the column.

The matric potential and hydraulic conductivity calculations involve only parallel computations. They both depend on the calculation of the type:

```
result := a * A^b
```

where 'a' and 'b' are scalars and 'A' is a parallel array (which is the same for both). A parallel version of this was calculated by:

```
temp := ln( A )
result := a * exp( b * temp )
```

where 'temp' is a temporary variable.

The savings in time by only calculating the natural logarithm once and doing the calculations in parallel will be considerable as the model is extended to more dimensions. The derivatives are also easily available through parallel operations as the change between layers can be obtained easily through the 'shift' operator and the thickness of each layer is stored in a parallel array. The same is true of the fluxes. Computation of thermal parameters involved mixing some scalar values with information from various points in parallel arrays and storing the results back in the parallel arrays. Here is where the special purpose routines were used. Simple 'get' and 'put' routines were written to get/put a value out of/into a user specified row and column in a parallel array. This could be done quickly using the special capabilities of the MPP architecture. Moreover, as the needed scalar computations were being done, parallel operations could be performed concurrently in the Array Unit.

Minimizing the Program Execution Time by Using the Capabilities of the MPP Through MPP Pascal

The multiprocessing capabilities of the MPP are easily available through MPP Pascal. Scalar calculations are performed in the Main Control Unit. This is a special purpose microcoded 16-bit processor which has a 16 bit hardware multiplier. Parallel calculations are performed in the Array Unit with 16384 specially designed processors. The scalar Main Control Unit calculations and the parallel Array Unit calculations are done simultaneously except when the Main Control unit is expecting a scalar result from the Array Unit. This would be the case, for example, when doing a maximum, minimum or sum operation on a parallel array. MPP Pascal produces a code which runs in the Main Control Unit and makes calls to library and special purpose routines which run in the Array Unit. There is also a call queue which enables the Main Control Unit to stack its calls (including register transfer data) to the Array Unit. These calls may be stacked up to 15 deep. Thus a parallel operation, such as an assign, in MPP Pascal translates to a single call by the Main Control Unit to the Array Unit to begin its processing with its own processors. The Main Control Unit is then free to do either scalar calculations or send another parallel operation request to the Array Unit. By recognizing that the Main Control Unit is a serial processor it becomes apparent that sending requests to the Array Unit to perform parallel operations and then doing scalar operations in the Main Control Unit allows the scalar and parallel calculations to be done at the same time.

This feature of the MPP was used extensively in the boundary condition calculations to reduce program execution time. It proved to be the single most important tool for reduction of program running time. Other techniques involved setting up masks at initialization time and reusing them instead of regenerating them with 'WHERE' statements, and also the use of temporary stores for the results of natural logarithm and exponent functions which were to be used in more than one calculation.

A 24 hour simulation on the MPP with 14 layers used 30 seconds of computer time, whereas the identical simulation on the IBM 3081 serial machine used 4 seconds. Adding more layers to the MPP model would use virtually the same amount of time, whereas the execution time on the serial machine is approximately linear with the number of layers. We expect, then, that the break even point is approximately 115 layers. Modelling an entire catchment could easily require ten times this number, so we expect that the parallel architecture of the MPP will provide significant savings in computer time over a similar model on a serial machine.

CONCLUSION

We have coded a one-dimensional hydrological model of the surface energy and moisture balance and moisture flow in the unsaturated zone, as a precursor to a complete catchment model.

By comparing to an identical model on an IBM 3081 serial machine, we have shown that it is feasible to use the MPP for numerical models such as this one, and that the parallel architecture makes such calculations more efficient when the physical model includes modelling the same processes at many different points in space.

ACKNOWLEDGEMENT

Two of the authors, J. E. Devaney and A. R. Irving, were partially supported by NASA contract NAS5-28200.

REFERENCES

1. Booth, A.D., Numerical Methods, Academic Press, New York, 1957.

2. Camillo, P.J., R.J. Gurney and T.J. Schmugge, A Soil and Atmospheric Boundary Layer Model for Evapotranspiration and Soil Moisture Studies, Water Resour. Res., 1983, 19, pp. 371-380.

3. Clapp, R.B. and G.M. Hornberger, Empirical Equations for Some Soil Hydraulic Properties, Water Resour. Res., 1978, 14, pp. 601-604.

4. de Vries, D.A., Heat Transfer in Soils, Heat and Mass Transfer in the Biosphere, Scripa, Washington, D.C., 1975.

5. Eagleson, P.S., Dynamic Hydrology, McGraw-Hill, New York, 1970.

6. Freeze, R.A., Three-Dimensional, Transient, Saturated-Unsaturated Flow in a Groundwater Basin, Water Resour. Res., 1971, 7, pp. 929-941.

7. Lin, J.D., On the Force-Restore Method for Prediction of Ground Surface Temperature, J. Geophys. Res., 1980, 85, pp. 3251-3254.

8. Philip, J.R. and D.A. de Vries, Moisture Movement Porous Materials Under Temperature Gradients, EOS Trans. AGU, 1957, 38, pp. 222-228.

9. Teddington, A., Modern Computing Methods, Philosophical Library, NY, 1958.

PHYSICS SESSION I

Session Chair: Prof. L.R. Owen Storey
Stanford University

N87-26535

PHASE SPACE SIMULATION OF COLLISIONLESS STELLAR SYSTEMS ON THE MASSIVELY PARALLEL PROCESSOR

Richard L. White
Space Telescope Science Institute
Baltimore, MD

ABSTRACT

A numerical technique for solving the collisionless Boltzmann equation describing the time evolution of a self-gravitating fluid in phase space has been implemented on the Massively Parallel Processor (MPP). The code performs calculations for a two dimensional phase space grid (with one space and one velocity dimension). Some results from calculations are presented.

The execution speed of the code is comparable to the speed of a single processor of a Cray-XMP. Advantages and disadvantages of the MPP architecture for this type of problem are discussed. The nearest-neighbor connectivity of the MPP array does not pose a significant obstacle. Future MPP-like machines should have much more local memory and easier access to staging memory and disks in order to be effective for this type of problem.

Keywords: Stellar Dynamics, Phase Space, Galaxies, Star Clusters, Parallel Processing, N-body Techniques

INTRODUCTION

The dynamics of gravitating systems are of great interest because of their application to many astronomical systems such as clusters of stars, galaxies, and clusters of galaxies. These problems have two important characteristics which make them ideal for study using a parallel computer. First, the fundamental physics governing the evolution of the system is both simple and well-known: the force between any pair of bodies in the system is just determined by Newton's law of gravitation. Second, this simple law must be applied very many times during the course of each time step, and all of the force calculations could in principle be performed in parallel.

N-body Methods

Unfortunately, direct N-body methods, in which the force between every pair of bodies is calculated in each step, are not practical for large stellar systems such as galaxies. Galaxies have so many stars that the gravitational potential is very smooth, and gravitational encounters between individual stars are rare. Consequently, such systems are described as collisionless. The actual number of stars is far too large for direct N-body simulation ($N \approx 10^{11}$), and computations with feasible values of N ($\approx 10^4$) produce many more 2-body encounters (collisions) than occur in the real systems.

Most studies of the dynamics of collisionless stellar systems have used a modified N-body technique in which the density distribution is binned and smoothed before calculating the gravitational potential (*e.g.*, Ref. 1). The calculation can then include many more particles than direct N-body simulations ($N \approx 10^5$), and 2-body encounters are prevented because the potential is smoothed. Even this method, however, suffers from graininess due to the finite number of particles in each zone; this graininess manifests itself as a numerical "noise" in the calculation. It also limits the resolution which is achievable with a given number of particles. Another problem is that it is necessary to use a "softened" gravitation potential instead of the real $1/r$ potential to avoid serious instabilities. Sellwood (Ref. 1) has shown that this significantly changes even the linear behavior of the system.

Phase Space Methods

Numerical noise can be eliminated if the stellar system is treated as a fluid in phase space. The evolution of such a system is described by the collisionless Boltzmann equation. A Japanese group has done some calculations using this approach (Refs. 2–5). Their results are very interesting and indicate that the numerical noise in the N-body methods may have led to spurious results. For example, Nishida (Ref. 5) found that the bar instability of a thin stellar disk can be suppressed by the presence of a small bulge component; previous N-body calculations had led to

PRECEDING PAGE BLANK NOT FILMED

the opposite conclusion, apparently because their resolution was inadequate.

This paper reports on the implementation on the Massively Parallel Processor (MPP) of a new numerical technique for phase space techniques. The technique is briefly described, some results are presented, the MPP implementation is discussed, and some improvements are suggested for future MPP-like machines which would make them more effective for this type of problem.

THE NUMERICAL TECHNIQUE

The numerical method described here was developed by the author and P. R. Woodward (Refs. 6, 7). Phase space is divided into an Eulerian (fixed) grid. This grid is two-dimensional (2-D) for spherical systems without angular momentum, 3-D for spherical systems with angular momentum, or 4-D for disk systems. The grid need not be rectangular; for the disk problem, for example, a polar grid can be used. For each zone of the grid, we store the mean density in the zone and all the first order moments in the phase space coordinates. Thus, for a 2-D grid, there are 3 moments: the mean density, the x-moment, and the v-moment. The moments in a zone imply a unique distribution of density within that zone which is linear in all coordinates. For a 2-D grid, the density is $f(x,v) = \langle f \rangle + f_x x + f_v v$, where $\langle f \rangle$ is the mean density and f_x and f_v are proportional to the x and v moments.

A version of this scheme using second order moments has also been implemented; however, this scheme requires much more memory than the linear scheme, especially for higher dimensional problems. The limited memory of the MPP made it desirable to use only linear moments.

A time step consists of calculating new values for the moments in each zone. The x and v motion is split into two half steps; by taking steps in the order x-v-v-x, second order accuracy in the time step is achieved. Fluxes of each moment across the zone boundaries are computed from the acceleration and velocity of material in the zone. The moments are updated using the fluxes so that mass and momentum are conserved during the time step.

RESULTS OF CALCULATIONS

The figures show the results for two calculations of the gravitational collapse of a 1-D, spatially periodic system consisting of infinite sheets of stars. For both examples, the spatial period is 10 Jeans lengths, the mean space density is 1, the Gaussian velocity dispersion is 1, and the 2-D phase space grid is 128 × 128.

The initial conditions differ for the two examples. The example in Figure 1 has a density perturbation of the form $\delta\rho = A \cos 2\pi x/L$, where L is the length of the grid and $A = 0.01$. The example in Figure 2 has $\delta\rho = A[\cos(2\pi x/L) + 3\cos(6\pi x/L)]$.

The figures display the evolution of the system with time. One spatial period is shown in each figure. For each time, a contour plot of the phase space density is shown, with velocity on the vertical scale and position on the horizontal scale. The contours are logarithmically spaced.

At the beginning of the calculation, the material is concentrated near zero velocity and is almost uniformly distributed in space. The initial perturbation causes material to feel a gravitational acceleration toward the center. This increases (decreases) the velocity of material to the left (right) of center, so that it moves up (down) in the figure. The higher velocity material then moves toward the center, which leads to the winding up of the original phase space distribution. As time passes, the original material becomes so tightly wound that it is no long resolved by the grid; it then appears smooth.

These two calculations reveal the interesting feature that the final phase space density distribution is not identical for the two collapses, so that there is some "memory" of the initial perturbation. This is of interest in understanding the formation and evolution of galaxies; it hints that buried in the present-day structure of galaxies there may still be information about the conditions under which galaxies were formed.

IMPLEMENTATION ON THE MPP

The implementation of this numerical scheme on the MPP is relatively straightforward. The 2 dimensional phase space grid is mapped directly to the 128 × 128 array of processors in the MPP, with one zone per processing element. The nearest neighbor connectivity of the MPP is precisely what is required for the flux-based method described above. The programmable boundary conditions for the MPP array allow the boundaries of most problems to be treated easily.

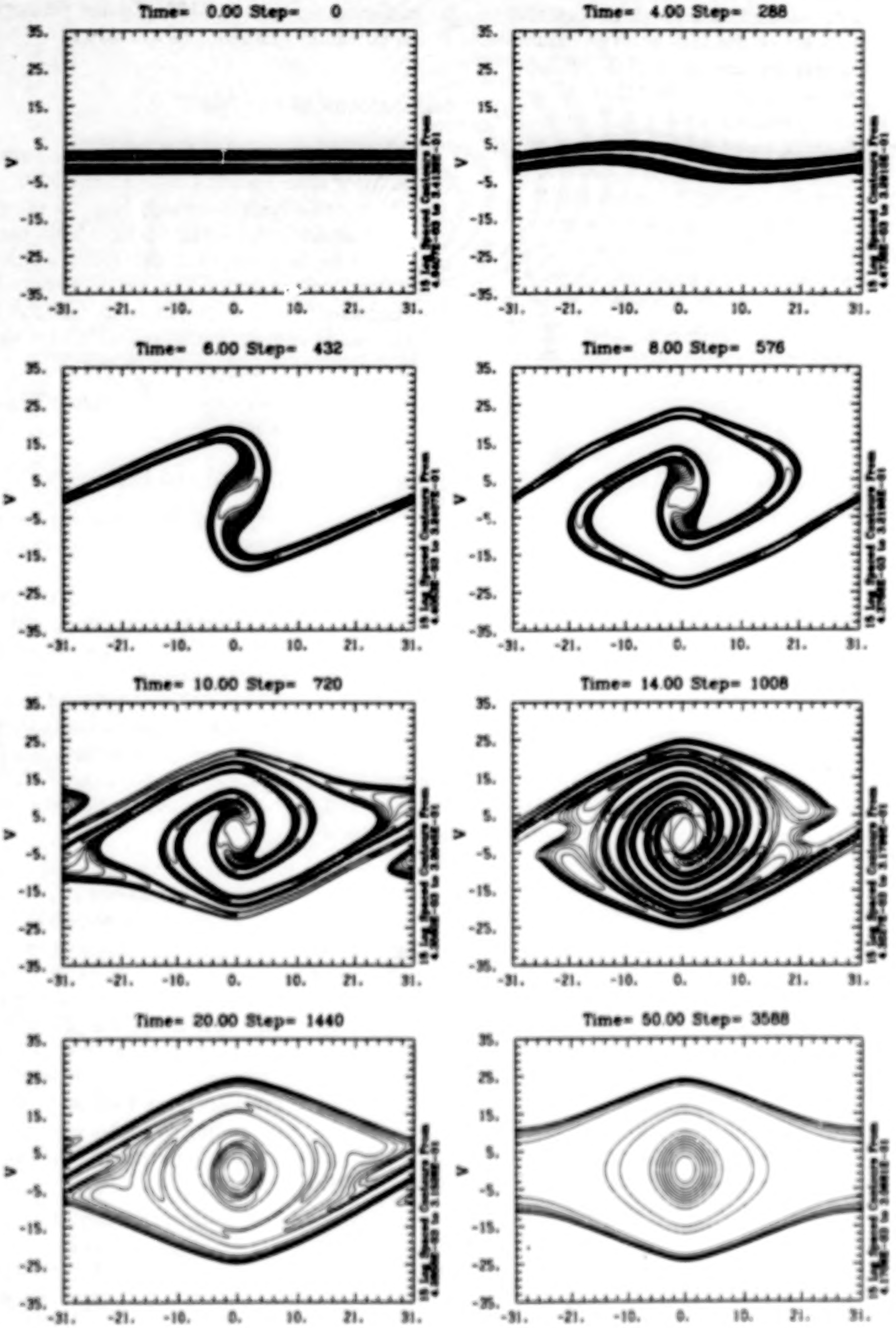


Figure 1. Evolution of phase space density for Example 1 (see text for details).

The MPP architecture is very effective for problems like the one described here which require intensive floating point computations. The MPP is divided into a scalar processing unit (the MCU) and a parallel processing array; the MCU places requests for array operations into a call queue. The time for a floating point add or multiply in the array is long compared to any computations which must be formed in the MCU; consequently, the MCU keeps the call queue full, and the array is working continuously. The array-side code consists of primitives which can (in principal) be fully optimized; the user code resides in the MCU and need not be optimized at all except in its organization of calls to the array, because the execution time of the MCU code has negligible effect on the total MPP time required for the calculation. The net effect of all this is that execution time on the MPP is very close to what one would calculate from simply counting the number of floating point operations and multiplying by the time per operation.

Performance of the MPP Code

The execution speed of the MPP code is about 50 times faster than a VAX 8600. This is comparable to the speed of the same method on a single processor of a Cray-XMP/48, even though the much greater vector length of the MPP (16384 versus 64) requires about 5 times more multiplies than the Cray within the inner loop. For a grid with uniform spacing, at each time step there are many coefficients which need be calculated only once for each row of the grid. In the Cray version of the code, the short vector length allows these coefficients to be calculated outside the inner loop, but on the MPP the coefficients must be re-calculated for every zone. This greatly increases the number of operations within the inner loop, so that even though the MPP can perform more multiplies per second than the Cray, the speeds of the machines are similar for this scheme.

For a problem using a grid with non-uniform spacing, the MPP program would be faster than the Cray by a factor of 3 or more, because nearly all the calculations would have to be performed within the loop by both the Cray and the MPP.

It appears that a 3 or 4 dimensional phase space calculation will also run several times faster on the MPP than on a Cray. However, higher dimensional problems require much more memory than is available within the MPP array. They can be implemented on the MPP if some limitations which currently exist can be alleviated.

Limitations of the MPP

The code development time was much greater on the MPP than on the Cray, and the MPP program is more difficult to modify (*e.g.*, to use grids larger or smaller than 128 × 128). This can be attributed to limitations of the MPP which fall into two classes: those which are determined by the hardware (which probably cannot be fixed for the current machine), and those which are determined by software (which can be fixed).

Hardware Limitations – Hardware limitations of the MPP include:

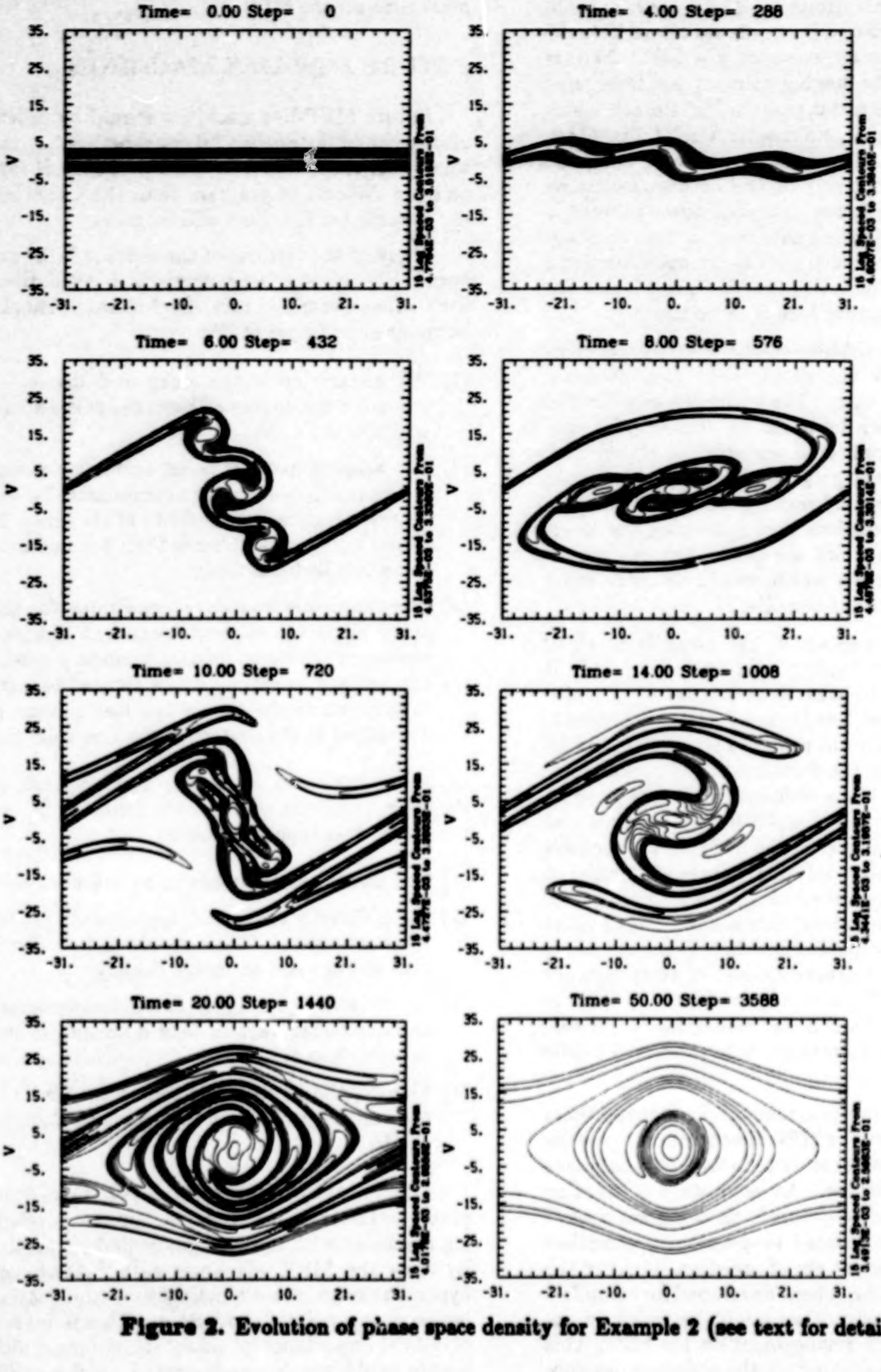
(1) the lack of direct I/O facilities, and

(2) the severe shortage of local memory for each processing element in the array.

The MPP is slowed significantly compared to a Cray if many intermediate results must be stored on disk. On the MPP, all input and output (I/O) is performed through the front-end VAX. The rate at which the VAX can respond to MPP I/O requests is often the limiting factor in the performance of the program. In contrast, the Cray has dedicated, high speed disks so that the I/O usually has little effect on the execution time for the program.

Each node in the array has only 1024 bits of local memory. Some of this memory is required for the system, leaving room for about 25 32-bit floating point numbers for all variables and temporary storage. This local memory is so small that it is difficult to implement a modestly complicated algorithm such as the one discussed in this paper. It is necessary for the programmer to manage memory very carefully, with temporary variables being constantly re-used within modules. This sort of coding is highly prone to errors, and the resulting bugs are difficult to find and eradicate.

Even using the staging memory, which contains a much larger reservoir of storage, some algorithms will be very inefficient. The need for numerous temporary variables may force one to the unpleasant prospect of having to roll temporary variables in and out of staging memory frequently during each time step.


Time= 0.00 Step= 0

Time= 4.00 Step= 288

Time= 6.00 Step= 432

Time= 8.00 Step= 576

Time= 10.00 Step= 720

Time= 14.00 Step= 1008

Time= 20.00 Step= 1440

Time= 50.00 Step= 3588

Figure 2. Evolution of phase space density for Example 2 (see text for details).

Software Limitations – The hardware problems are exacerbated by shortcomings of the software that currently exists on the MPP. Neither disk I/O nor the staging memory are integrated into the high-level language MPP Pascal, which was used for the implementation of the phase space method. The existing utility routines do not make use of many of the hardware capabilities of the MPP. For example, the hardware is capable of transferring data to and from the staging memory in parallel with array operations and with no interaction with the VAX; this is not implemented in current high level routines.

Software limitations of the MPP Pascal compiler also make the shortage of local memory harder to deal with. The compiler uses far too many temporary locations in memory for any reasonably complicated assignment statements. Consequently, the programmer is required to do the compiler's job of breaking each equation down into binary operations and managing the intermediate results which are generated. This is an error-prone process which would be much more efficiently left to the compiler.

The local memory of the array is so small (when used for floating point numbers) that it might be best to view it as similar to the general purpose registers in many scalar computers. There are usually not enough registers to hold all of one's data for the duration of the problem; instead, the data is stored in main memory (staging memory, for the MPP) and is brought in a piece at a time for processing. In the MPP, approximately four floating point multiplies can be performed in the time required to move a 32-bit number from the stager to the array; this is slow compared to many scalar computers, but it still may be short enough for the "register model" of array memory to be viable. The time required for this transfer becomes negligible if it can be carried out in parallel with array operations, as is permitted by the hardware.

Requiring the programmer to manage these data transfers on the MPP is equivalent to requiring the programmer to write in assembly language on a scalar machine. An advantage of treating array memory as registers is that a great deal of effort has been devoted to developing compilers which make efficient use of registers. Many of the techniques that have been developed for compilers on scalar computers thus might be fruitfully applied to memory management on the MPP. This would dramatically reduce the software development time on the MPP.

FUTURE MPP-LIKE MACHINES

Future MPP-like machines should be orders of magnitude faster than future short vector machines like the Cray. They will probably always be more difficult to program than the Cray, but the rewards for the effort will be great.

Many of the features of the current MPP are beautifully suited to large numerical calculations like the one described here. Such features should be retained in future MPPs:

(1) The separation of the array and the scalar processor allows very efficient use of the array, as discussed above.

(2) The staging memory is an extremely useful device which does much to compensate for the nearest neighbor connectivity of the array. It allows problems with more than 2 dimensions to be handled effectively.

(3) Programmable boundary conditions for the array allow the efficient treatment of many problems. Some additional boundary conditions might be useful (*e.g.*, a twisted connection in which the top of the last column is connected to the bottom of the first column.)

On the other hand, there are some aspects of the MPP architecture that could definitely be improved for this type of problem:

(1) The local memory needs to be much bigger.

(2) Some directly connected, high speed I/O devices are needed. These should include direct disk storage and an image display.

(3) As the array gets bigger, it will be important to have faster ways to load data into it and to shift data across it.

(4) Finally, the hardware is important, but the accompanying software development must not be forgotten.

Much of the current research in parallel computer architectures is directed toward developing machines with more sophisticated connectivity than the MPP. Machines with butterfly or hypercube connections allow easy communication between any pair of processors. This is indeed of critical importance for many algorithms, which simply could not be implemented on the MPP.

For example, the most promising N-body methods organise the particles into a binary tree and communicate information about masses and positions along the branches of the tree. The MPP would be very poorly suited for such an algorithm.

On the other hand, the MPP is very well suited to large numerical problems involving fluid flow, in which the connectivity of the physics underlying the calculation is usually local. It is also efficient for image processing, the task for which it was designed. It would be very premature to conclude that the nearest neighbor architecture of the MPP should be abandoned. The simplicity and easy expandability of MPP-like arrays should make such machines attractive as number-crunching engines for the foreseeable future.

CONCLUSIONS

Phase space techniques have several advantages over N-body calculations for collisionless stellar systems: there is no need for a "softened" gravitational potential; there is no numerical noise; and the resolution can be made higher by choosing the grid appropriately. It is also much easier on machines like the MPP to implement phase space fluid schemes than to implement efficient N-body schemes.

Phase space techniques do have one significant disadvantage compared to particle methods: they require more computational time and memory for multi-dimensional problems. For example, a 3-D problem requires a 6-D phase space calculation. Since most of phase space is empty, much of the computing time in such a problem would be wasted. By comparison, a particle calculation represents the ultimate Lagrangean grid calculation: all of the computational effort is expended where the matter is located.

The MPP is very well suited to fluid schemes, though some suggestions for improvements in hardware and software have been made. The most important change for future MPPs would be the addition of much more local memory for each node.

REFERENCES

1. Sellwood, J. A. 1983, *J. Comp. Phys.*, **50**, 337.
2. Fujiwara, T. 1981, *Pub. A. S. J.*, **33**, 531.
3. Watanabe, Y., *et al.* 1981, *Pub. A. S. J.*, **33**, 541.
4. Nishida, M. T., *et al.* 1981, *Pub. A. S. J.*, **33**, 567.
5. Nishida, M. T. 1986, *Ap. J.*, **302**, 611.
6. Woodward, P. R., and White, R. L. 1986, *J. Comp. Phys.*, in preparation.
7. White, R. L. 1986, in proceedings of the Princeton meeting on *The Use of Supercomputers in Stellar Dynamics*, ed. P. Hut.

N87-26536

PARTICLE SIMULATION OF PLASMAS ON THE MASSIVELY PARALLEL PROCESSOR

I. M. A. Gledhill   L. R. O. Storey
Space, Telecommunications and Radioscience Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

ABSTRACT

Particle simulations, in which collective phenomena in plasmas are studied by following the self-consistent motions of many discrete particles, involve several highly repetitive sets of calculations that are readily adaptable to SIMD parallel processing. We describe a fully electromagnetic, relativistic plasma simulation for the MPP. The particle motions are followed in $2\frac{1}{2}$ dimensions (two spatial and three velocity) on a 128 × 128 grid, with periodic boundary conditions. The two-dimensional simulation space is mapped directly onto the processor network; a Fast Fourier Transform is used to solve the field equations. Particle data are stored according to an Eulerian scheme, i.e., the information associated with each particle is moved from one local memory to another as the particle moves across the spatial grid.

The method is applied to the study of the nonlinear development of the whistler instability in a magnetospheric plasma model, with an anisotropic electron temperature. The wave distribution function is included as a new diagnostic to allow simulation results to be compared with satellite observations. Since the physics of self-gravitating systems is quite similar to plasma physics, incorporation of free-space boundary conditions and alteration of the field equations enable our code to be used for the study of density waves in galaxies.

Keywords: Particle Simulation, Plasma Physics, Stellar Dynamics, Parallel Processing.

1. INTRODUCTION

The research work described in this paper belongs to a relatively new category: the exploratory use of parallel processors for the particle simulation of rarefied media. By a 'rarefied' medium we mean one in which the collisional mean free paths are not necessarily small on the scale of the systems considered, with the result that it cannot always be approximated satisfactorily as a continuous fluid. Plasma, which is the main constituent of the Universe, is a prime example. Among the very diverse phenomena that can occur in a plasma, some of the most interesting are highly nonlinear, and therefore difficult to analyze theoretically. For the plasma theorist, a powerful alternative and complement to analytic study is the use of particle simulation in a computer.

'Particle simulation' is the generic term for computational procedures in which a medium is represented in the computer as an assembly of discrete interacting particles. In a plasma the particles are ions and electrons, interacting through the electric and magnetic fields that they themselves create. At each time step in a simulation, the computer has two distinct tasks to perform: it must update the positions and velocities of the particles, taking account of their accelerations due to the fields, and it must update the electric and magnetic fields, the sources of which are, respectively, the charge density calculated from the positions of the particles, and the current density calculated from both their positions and their velocities.

Now, even a modest volume of plasma may contain a very large number of particles: in Earth's ionosphere, for instance, there are typically 10^{12} electron-ion pairs per cubic metre. Computer simulations necessarily involve much smaller numbers, not more than a few times 10^6 for present-day single-processor computers ('uni-processors'), so each of the simulation particles actually stands for very many particles in the real world; they are sometimes called 'superparticles' for this reason.

Besides the reduction in the number of particles, other simplifications are often made in order to reduce the demand for computing time. The commonest is to reduce the dimensionality, by considering systems in which all physical quantities vary in only one or two dimensions. This simplification can be very helpful, provided that it is authorized by the symmetry of the problem, but otherwise it is objectionable because it makes the simulation unrealistic. Another common way of simplifying plasma simulations is to ignore electromagnetic radiation by as-

PRECEDING PAGE BLANK NOT FILMED

N87-26536

suming that the speed of light is infinite. However, the real world is three-dimensional and the speed of light is finite, so most of the outstanding problems in plasma physics that are amenable to particle simulation will ultimately need to be tackled in three dimensions (3-D) using a fully electromagnetic (EM) code.

Probably the most advanced EM particle code in existence is the TRI-dimensional STANford code TRISTAN, written in assembler language for the Cray-1 computer; see Ref. 1 for an account of an earlier version of this code. It follows the motion of about 5×10^6 particles in a cubical volume divided up into 128^3 cells, i.e., each side of the cube is divided into 128 units. These numbers are not extravagant: if anything, they err on the side of modesty, and codes with more particles and cells are likely to be required in the future. Already, however, the preliminary tests of TRISTAN have revealed a severe problem of computer usage. Though the assembler-language code has been carefully optimised, a single time step requires 2-3 minutes of CPU time on the Cray-1, and a typical simulation requires 500-1000 steps. Difficulty in obtaining the requisite amount of computer time has already set back several research programs where the use of TRISTAN was envisaged. This is just one instance of a critical situation which is now widely recognised, namely that advanced particle simulation is up against a barrier due to the speed limitations of uni-processors.

Other advanced approaches to 3-D plasma simulation exist, namely the statistical methods involving numerical integration of the collisionless Boltzmann equation or of the Fokker-Planck equation. Formally, they are equivalent to following the motion of a fluid in a 6-D space, which has three dimensions of velocity as well as the three dimensions of position. For a given number of cells in position space, however, these methods demand even greater computing speed.

The prospects for large increases in the speed of uni-processors are not encouraging: fundamental physical constraints on VLSI technology are expected to limit them to factors of less than 100. For larger increases, we must look to multi-processors, i.e., computers consisting of multiple processors arranged in parallel architectures, such as the MPP.

Though the MPP was designed originally for processing image data from the Landsat satellites, its architecture, involving a large number of simple processors with nearest-neighbor connections, is well suited to the particle simulation of rarefied media, and incidentally to fluid simulation as well (Ref. 2). The motion of particles or fluid in a given spatial cell is determined only by conditions prevailing inside that cell, and at its boundaries with neighboring cells. (At least, this is the case so long as we refrain from making the approximation in which disturbances propagate across the array of cells instantaneously). Hence problems concerning such motion can be mapped readily onto simple arrays of processors in which direct connections exist only between nearest neighbors.

In the present program of research, our initial aims were to gain enough experience in the use of the MPP to be able to answer the following questions:

- Which of our plasma simulation problems can be solved on this type of multi-processor?
- Is there a significant gain in speed, compared with plasma simulations on a uni-processor?
- Could the capability of the MPP for plasma simulation be improved in any simple way?
- Would any other type of multi-processor be better suited to our problems, and if so which?

In sum, we wished to investigate the potentialities and limitations of massively parallel processors for plasma particle simulation.

Shortly after the investigation began, however, its scope was extended to include particle simulation of problems in stellar dynamics, this in collaboration with Dr. Bruce Smith of NASA Ames Research Center. Stellar dynamics is very similar to plasma electrodynamics in respect of its basic physics. Both plasmas and stellar systems are examples of rarefied media, and problems concerning either of them are cases of the classical N-body problem with inverse square law interactions. Each, of course, also has its specificity: stellar dynamics involves long-range interactions, from which plasmas are exempt because of Debye shielding; plasma electrodynamics involves magnetic fields, which have no counterpart in non-relativistic stellar dynamics. Nevertheless, the two fields have much in common, so naturally there are close similarities between the methods of particle simulation that have been developed for each of them. For the same reason, many possibilities exist for cross-fertilization between them, and for joint efforts to solve common difficulties. These considerations led us to extend the scope of the program in this way.

The present paper is an account of the work performed during the first year. Section 2 outlines the two physical problems, one in plasma electrodynamics and the other in stellar dynamics, that have been chosen for simulation, and how we are approaching them; for simplicity, both simulations are in two spatial dimensions instead of three. Section 3 describes the numerical techniques used for the plasma simulation, and Section 4 the modifications required for the gravitational simulation. Section 5 indicates how these various algorithms are being implemented on the MPP, while the current status of the work is described in Section 6. Finally, in Section 7, we draw some provisional conclusions and sketch our plans for the future, in both the short and long terms.

2. PHYSICS PROBLEMS

Recapitulating, our immediate objectives are to program the MPP to solve, by particle simulation, a significant problem in each of the two fields mentioned above, namely stellar dynamics and plasma electrodynamics.

The simpler of the two problems is the one in plasma electrodynamics, which concerns the nonlinear development of the Doppler-shifted electron gyroresonance instability, or 'whistler instability' for short. It is a very suitable problem for solution on the MPP, for several reasons. Firstly, it is one in which only the electrons are involved, while the ions can be treated as an inert neutralising background. The simulation algorithms are simpler if only one type type of particle needs to be represented. Secondly, the whistler instability is of the velocity-space variety, so it can occur even in a spatially uniform plasma. By assuming uniformity as an initial condition, we should succeed in sharing the computing load evenly between the different processors. Finally, it is an electromagnetic instability, which means that field disturbances originating in any one spatial cell propagate to the other cells at finite velocities. Such propagation can be modeled readily on the MPP, in which each processor is connected only to its four nearest neighbors. Despite its apparent simplicity, the nonlinear evolution of the whistler instability has not been simulated before in the conditions in which we intend to do so. This problem has applications both to space and to fusion plasma physics.

To the best of our knowledge, all previous simulations of the whistler instability have been one-dimensional. The most recent, by Bharuthram and Baboolal (Ref. 3), were in several other respects quite similar to those that we are proposing. Thus, for instance, these authors used a fully electromagnetic code, gave the electrons a bi-Maxwellian velocity distribution at the outset, and followed the growth of the instability into the nonlinear regime. They obtained a variety of interesting scientific results: for instance, the wave mode that, during the initial linear phase of the instability, grew most rapidly, died out in the post-saturation phase, and ultimately was replaced by a slower-growing mode which dominated the wave spectrum at the end of the simulation.

Using our MPP code, we intend to study the nonlinear evolution of the whistler instability under much the same conditions, except that our simulations will be performed in $2\frac{1}{2}$-D (i.e., in two spatial dimensions, but with all three components of velocity). We expect that the increase in dimensionality will lead to qualitatively new scientific results: in particular, we anticipate that waves will be generated over a broad frequency band, with their wave normal directions spread widely around the direction of the steady magnetic field. We shall study how the characteristics of the wave field in the nonlinear regime depend on the initial plasma conditions.

For these simulations, the code will have to be enhanced with sophisticated diagnostics, especially for the fields. The fact that it already includes a two-dimensional FFT should be very helpful in this respect, enabling wave power spectra, dispersion relations, temporal autocorrelation functions, etc., to be derived by existing techniques, developed previously for serial processors.

Additionally, we propose to employ a new diagnostic procedure, hitherto used only for analysing whistler-mode wave data from satellites in the Earth's magnetosphere. This is the so-called 'wave distribution function' (WDF), which specifies how the wave energy density is distributed with respect to frequency and to wave normal direction (Refs. 4–7). The ready availability of this diagnostic will, in the future, facilitate comparisons between the results from numerical simulations of the whistler instability, and observations of natural whistler-mode wave fields generated by the same instability in space.

In stellar dynamics, we shall investigate the stability of an axisymmetric system (galaxy or protoplanetary nebula), albeit that problems of this type have been simulated previously on conventional computers. The interest of repeating one of these simulations on the MPP is that it will introduce some of the difficulties that are bypassed in the whistler instability problem. Although, again, the dynamics involve only one type of particle, the absence of a neutralising background of particles of another type means that the inter-particle forces are not shielded in any way, so they are truly long-range. For this reason, all physically significant systems are spatially non-uniform. Finally, the times required for gravity to propagate across such systems are very small compared with the time scales characterising their dynamics (e.g., the rotation period of a galaxy), so little accuracy is lost and much computing time saved by assuming that the propagation is instantaneous. Thus, besides the physical interest of this problem, its simulation on the MPP will have interesting computer science aspects.

3. NUMERICAL TECHNIQUES

Details of the particle-mesh method may be found in the standard texts on plasma simulation (e.g., Refs. 8–10). We summarise here the techniques chosen for the MPP code.

The purpose of the simulation model is to follow the self-consistent dynamics of fields and particles for several thousand time steps. Fields, densities and other spatial functions are stored as values at points on a grid. Initially, our simulation is restricted to two spatial dimensions for simplicity. The motion of a large number of particles (of the order of 10^5 to 10^6), each representing many particles in the real world, is followed through successive time steps.

Initially, particles are arranged in positions (x, y) that result in the required plasma configuration. For our present purposes, uniform density is specified. Particles are assigned random velocities (v_x, v_y, v_z) according to a Gaussian distribution with suitable thermal and drift velocities.

As is usual in numerical simulations, point particles are replaced by clouds of charge to prevent impulse-like interactions which give rise to collisional effects (Ref. 11). Long-range particle potentials are retained in the Coulomb form to model collective effects, but a shape function

$$S(\mathbf{r}) = \frac{1}{2\pi a^2} \exp(-r^2/2a^2) \qquad (3.1)$$

is introduced so as to 'soften' the particle potential within the range $r < a$ (usually of the order of a Debye length).

In order to calculate the fields set up by the charges, it is necessary to obtain the charge and current densities ρ and $\mathbf{J}$ on the spatial grid. A simple algorithm is used which produces the same results as the Subtracted Dipole Scheme (Ref. 12). The charge density array is compiled from the superposition of a unit charge at each particle's nearest grid point (NGP), plus small contributions at its four neighboring grid points which reproduce the dipole moment of the charge relative to its NGP.

The electric and magnetic fields ($\mathbf{E}$ and $\mathbf{B}$) are governed by Maxwell's equations. With the inclusion of the shape factor, Ampère's and Faraday's laws become

$$\frac{\partial \mathbf{E}(\mathbf{r})}{\partial t} = c\,[\nabla \times \mathbf{B}(\mathbf{r}) - 4\pi \mathbf{J}(\mathbf{r}) * S(\mathbf{r})] \qquad (3.2)$$

and

$$\frac{\partial \mathbf{B}(\mathbf{r})}{\partial t} = -c\,\nabla \times \mathbf{E}(\mathbf{r})\,, \qquad (3.3)$$

where the asterisk denotes convolution and c is the speed of light. Gauss' law,

$$\nabla \cdot \mathbf{E}(\mathbf{r}) = 4\pi \rho(\mathbf{r}) * S(\mathbf{r}) \qquad (3.4)$$

is satisfied by the initial conditions and will continue to hold if the continuity equation is satisfied. However, microscopic inconsistencies between ρ and $\mathbf{J}$ arise due to the use of the mesh (Ref. 13). To prevent the resulting growth of noise in the fields, it is common to split the electric field into longitudinal and transverse components $\mathbf{E}_l$ and $\mathbf{E}_t$ (Ref. 9), using Gauss' law to obtain $\mathbf{E}_l$ and Faraday's law to deduce $\mathbf{E}_t$. It is thus necesary to solve both elliptic and hyperbolic equations in this system; the use of Fourier or Hartley transforms provides a simple solution, as well as helpful diagnostics.

The field equations are integrated in time using a leap-frog method with time step Δt. Fourier transforming (3.2), (3.3) and (3.4) and including the time differencing gives

$$\mathbf{E}_t^n(\mathbf{k}) = \mathbf{E}_t^{n-1}(\mathbf{k}) + c\,\Delta t\,\{i\,\mathbf{k}\times\mathbf{B}^{n-\frac{1}{2}}(\mathbf{k}) -$$

$$4\pi\,S(\mathbf{k})\,[\mathbf{J}^{n-\frac{1}{2}}(\mathbf{k}) - (\mathbf{J}^{n-\frac{1}{2}}(\mathbf{k})\cdot\mathbf{k}/k^2)\,\mathbf{k}]\} \qquad (3.5)$$

$$\mathbf{B}^{n+\frac{1}{2}}(\mathbf{k}) = \mathbf{B}^{n-\frac{1}{2}}(\mathbf{k}) - ic\,\Delta t\,\mathbf{k}\times\mathbf{E}_t^n(\mathbf{k}) \qquad (3.6)$$

$$\mathbf{E}_l^n(\mathbf{k}) = -4\pi\rho^n(\mathbf{k})\,S(\mathbf{k})\,\mathbf{k}/k^2 \qquad (3.7)$$

The superscripts denote the time step at which the field is known:

$$f^n \equiv f(t_0 + n\,\Delta t) \qquad (3.8)$$

The stability of the method is governed by a Courant-Friedrichs-Levy (CFL) condition:

$$\frac{(k_{\max}\,c\,\Delta t)^2}{4} < 1 \qquad (3.9)$$

where $k_{\max}$ is the highest spatial frequency occurring in the simulation.

Once the fields are known on the spatial grid, the forces on the particles may be calculated by interpolation. The relativistic equation of motion is

$$\frac{d\mathbf{p}}{dt} = q\mathbf{E} + \left(\frac{q}{m_0 c}\right)\frac{\mathbf{p}\times\mathbf{B}}{\gamma} \qquad (3.10)$$

where p is the relativistic momentum, m_0 is the rest mass, and

$$\gamma = 1/\sqrt{1 - v^2/c^2} \qquad (3.11)$$

A problem arises because of the appearance of the momentum in the vector product in equation (3.9). A well-established solution due to Boris (Ref. 14) is employed: the momentum is advanced using the electric field only for half a time step, then a rotation of the resulting intermediate momentum vector by the magnetic field is carried out and finally the second 'half-push' by the electric field is performed.

One of the advantages of applying the full relativistic treatment is that a top speed for both field and particle propagation is provided, and can easily be tailored so that the CFL condition (3.9) is obeyed.

Advancement of the particles' positions closes the main simulation loop, since the charge and current densities may be evaluated again.

4. GRAVITATIONAL SIMULATION

Particle-mesh simulation is a well-known technique in the investigation of the evolution of spiral galaxies (Ref. 8). Since the algorithms for gravitational simulations are so closely related to those for plasma, it seems worthwhile to construct them in parallel. The main differences between the programs are summarised here.

The gravitational potential $\phi(\mathbf{r})$ is used instead of the fields. It is governed by Poisson's equation

$$\nabla^2\phi(\mathbf{r}) = 4\pi G\,\rho(\mathbf{r}) \qquad (4.1)$$

where ρ is the mass density and G the gravitational constant. Solutions are obtained by convolving the Green's function for this equation, i.e., the single particle potential, with the particle number density function ρ/m_0. The point particle potential

$$\phi = -\frac{G\,m_0}{r} \qquad (4.2)$$

is replaced by the 'soft' potential

$$\phi_p = -\frac{G\,m_0}{\sqrt{r^2 + a^2}} \qquad (4.3)$$

so that shape factors do not appear explicitly in this formulation. The convolution is accomplished in Fourier transform space.

For modeling isolated galactic systems, free-space boundary conditions are imitated by confining the particles to 1/4 of the grid, a square of sides L, and truncating ϕ_p beyond $x = \pm L/2$ and $y = \pm L/2$

(Ref. 8). Although periodic images of the simulation plane are still produced by the fast Fourier transform, they can no longer influence each other across the empty buffer zone.

5. IMPLEMENTATION ON THE MPP

It will be seen from the above description that there are two main areas of the simulation algorithm with a high degree of parallelism: handling the fields and densities on the spatial grid, and processing the large number of particles moving though the space. This dichotomy appears in the data structure adopted for the simulation.

In the simplest approach, a spatial grid of 128×128 cells is mapped directly onto the two-dimensional array of processing elements (PE's), as illustrated in Figure 1. The local memory holds ρ, $\mathbf{J}$, $\mathbf{E}_\ell$, $\mathbf{E}_t$ and $\mathbf{B}$ associated with the corresponding grid point. The contents of the array unit (ARU) memory may be shifted across the network and wrapped around at the edges; extensive use is made of this feature in the fast Fourier transform for solving the field equations. Because it is a global method, the FFT has the disadvantage of requiring communication between distant processors in the network. An attractive alternative is to employ local methods of field solution, in which each processor would demand data from its nearest neighbours only (Ref. 15). However, the FFT is retained for the present as a useful general tool, since its numerical stability is well understood, and also it provides standard diagnostics.

The particle data structure presents a more interesting problem. Provision must be made for the information associated with the particles to move across the network in order to interact with appropriate data on the grid. Two alternative schemes, which may be used in some combination, exist.

In one disposition, information for a given particle is stored in the memory location corresponding to its position on the grid (Figure 1). Local calculations of densities and forces may then be made without communication beween processors. However, the packet of information must be moved when the particle travels to a new cell in the grid. This method has the disadvantages that memory overflow may occur in some locations, and that the workload is not distributed evenly among the processors unless the particle density is more or less uniform.

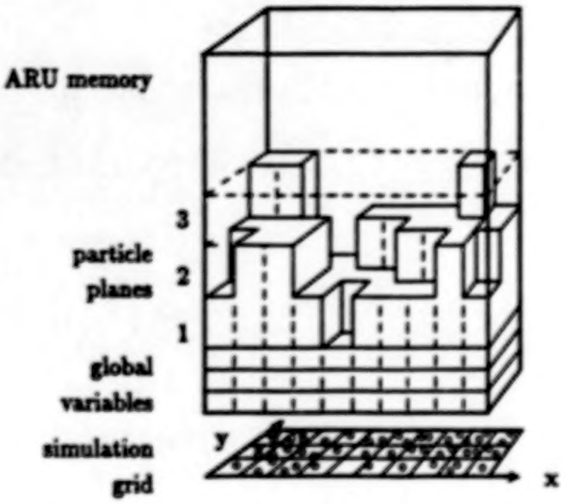


Figure 1. Mapping the particle data to memory. Each local memory contains data for particles in the corresponding cell in the simulation plane.

The alternative is to store particles in a uniform way in memory, in arbitrary locations. Particle and/or field data must then be transported across the network to a common processor when calculations involve both types. This scheme should prove useful when large density variations exist or when particles move though several cells per time step; Hoshino and Takenouchi (Ref. 16) proved a variation of it to be the more efficient scheme in a particle-particle molecular dynamics model on the PAX parallel processor.

In the case of an initially homogeneous plasma, however, large density differences are not produced except in extreme circumstances, and particles may be restricted from traveling more than one cell in a time step by setting the speed of light to this value. For our present purposes, therefore, we decided to use the first of the two schemes outlined above. The scheme also has the attraction of being a 'natural' mapping, and for this reason a variant of it was used in a 3-dimensional particle-mesh simulation of galactic gas dynamics on the ICL DAP by Johns and Nelson (Ref. 17).

Figure 2 shows a flow chart outlining the algorithm. Calculations are carried out in two major procedures, titled 'Particles' and 'Fields'. The only global parallel arrays that have to be carried between these are the variables required by both: ρ and $\mathbf{J}$ (the output of 'Particles' and the input of 'Fields'), and $\mathbf{E}_l$, $\mathbf{E}_t$ and $\mathbf{B}$ (the output of 'Fields' and the input of 'Particles'). These data are stored in the ARU throughout the main time loop.

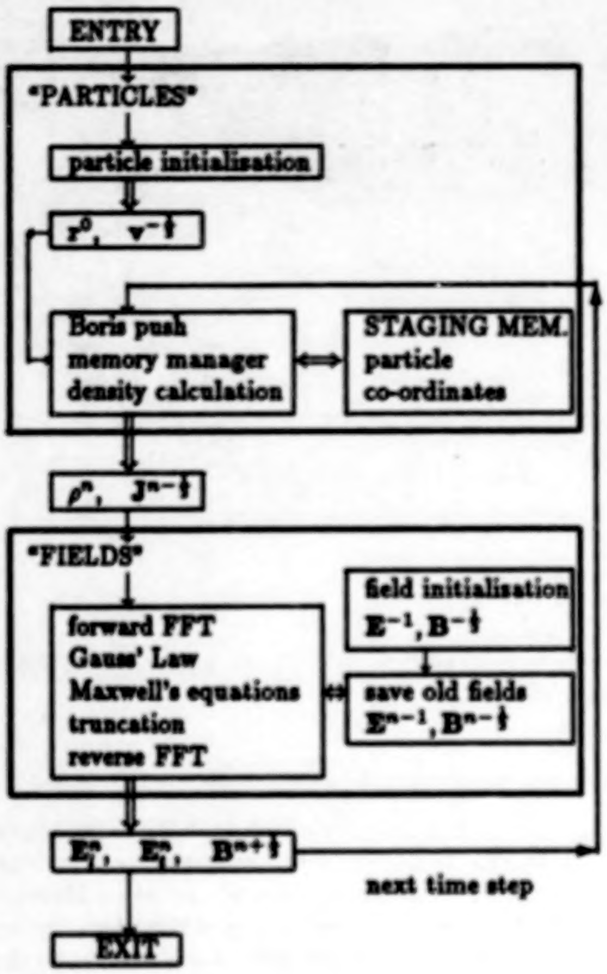

ENTRY
"PARTICLES"
particle initialisation
$\mathbf{r}^0$, $\mathbf{v}^{-\frac{1}{2}}$
Boris push
memory manager
density calculation
STAGING MEM.
particle
co-ordinates
ρ^n, $\mathbf{J}^{n-\frac{1}{2}}$
"FIELDS"
field initialisation
$\mathbf{E}^{-1}, \mathbf{B}^{-\frac{1}{2}}$
forward FFT
Gauss' Law
Maxwell's equations
truncation
reverse FFT
save old fields
$\mathbf{E}^{n-1}, \mathbf{B}^{n-\frac{1}{2}}$
$\mathbf{E}_l^n$, $\mathbf{E}_t^n$, $\mathbf{B}^{n+\frac{1}{2}}$
next time step
EXIT

Figure 2. Flow chart for the plasma simulation algorithm.

Particle data exist in the ARU only as local variables in the procedure 'Particles', and are stored in the staging memory (SM) during the rest of the execution. The procedure contains a loop which is executed once for each plane of particle data in the SM, as follows. A 'particle plane' is moved from the SM to the ARU. Values of the magnetic field and the total electric field are interpolated in each cell at the particle positions (x, y). (Operations are masked out in cells containing no particles). New velocity components are obtained by applying the Boris push, and the position is updated with the new velocity. The new co-ordinates (x, y) are tested to see whether the particle has moved to a new cell on the simulation grid, and a moving direction D assigned accordingly. The speed of light is normalised to one cell per timestep, so that no particle can move further than to one of its eight nearest neighbours; hence there are nine possible values of D, including the null move.

An inner loop is then carried out for each direction. The particles with appropriate D are copied into a workplane which is shifted to the new location. The densities ρ and $\mathbf{J}$ are updated with the new information. Distribution functions are accumulated when needed via a cascade sum. The workplane contents are then copied to the top of the stack of particles already moved. If full planes exist in the stack, they are sent to the SM. The next SM plane is then retrieved. After the last plane has been processed, the remaining particle planes are written out to the SM to clear the ARU of particle data.

In a fully electromagnetic program, the field solver requires nearly all the available ARU space. The 1-dimensional FFT that we use is based on a straightforward Cooley-Tukey algorithm (Ref. 18), and operates on 128 elements along either rows or columns.

Equations 3.5 to 3.7 are solved in transform space using the $\mathbf{E}_l$, $\mathbf{E}_t$, and $\mathbf{B}$ values from the previous time step. The shape factor and coefficients involving $\mathbf{k}$ are recalculated at each time step to minimise the space required for global variables. Transverse fields are truncated for $\mathbf{k}$ greater than some value specified by the user to meet the CFL condition 3.9. The spatial variation of one component of the field of a single particle is shown in Figure 3.

Particle initialisation is accomplished in 'Particles' by setting up co-ordinates in the ARU plane by plane. A random number generator based on a random bit-plane generator developed at NASA Goddard Space Flight Center is used to initialise velocities. Up to twelve uniformly distributed random numbers are added to produce a distribution function closely approximating a Gaussian. Field initialisation takes place through the inclusion of 'old' fields in Maxwell's equations: external fields are introduced in this way.

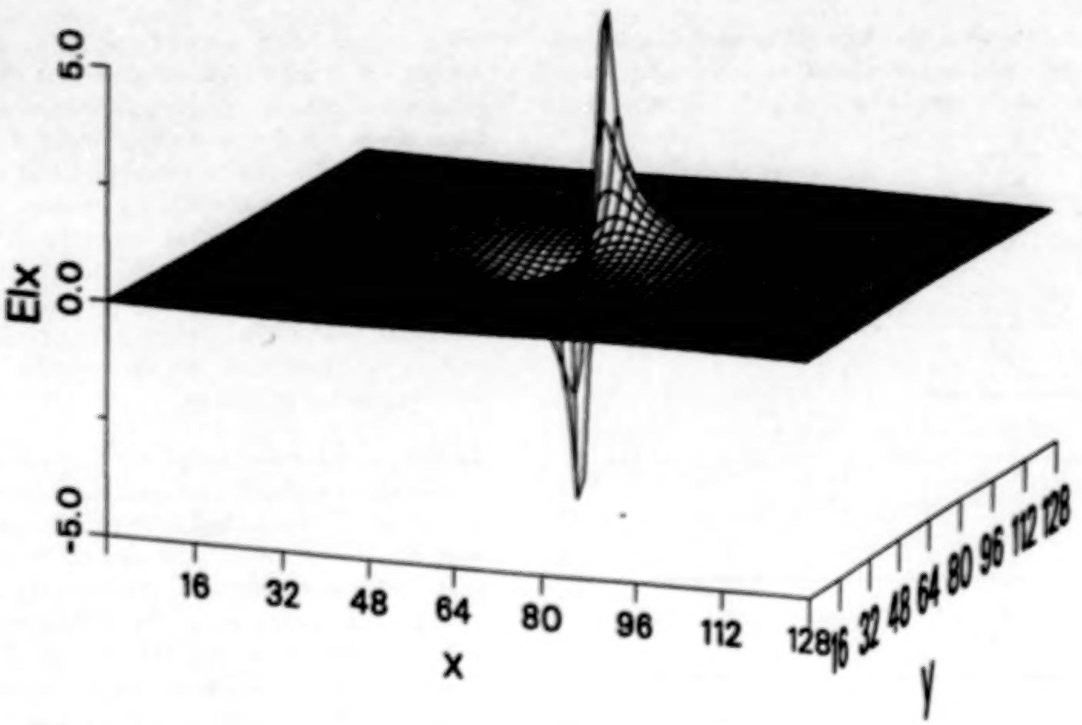


Figure 3. Spatial variation of the *x*-component of the longitudinal electric field due to a charge with radius 1 grid cell placed at the center of the grid. Units are arbitrary.

6. STATUS OF THE CODE

Considerable time has been spent in developing basic tools such as the FFT, the random number generator, input/output routines, etc. These are now in working order. The particle memory manager is complete and is being tested with the particle initialisation. The calculation section of 'Particles', which contains the interpolation, the Boris push, the distribution function calculation and the density accumulation, is in the final stages of development. The procedure 'Fields' awaits only the debugging of the transverse field section; preliminary results have been shown in Figure 3. Thus, as we go to press, the code appears to be close to final assembly.

7. CONCLUSIONS

Since we have not yet performed a complete simulation on the MPP, it would be premature to try to give any firm answers to the four questions raised in Section 1. Nevertheless, we feel that sufficient progress has been made to allow us to express opinions as to what the answers are likely to be. For instance, it is already clear that the two physics problems we have chosen, one in plasma physics and the other in stellar dynamics, can be simulated on the MPP without posing any difficulties of principle. The main practical difficulty is likely to be that of load balancing, i.e., of ensuring that the computational work load is shared evenly between all the processors in the array. In our present computational scheme, where the simulation domain is mapped directly onto the processor array, load balancing becomes difficult whenever the physical medium is inhomogeneous. Thus our stellar dynamics problem is more difficult, in this respect, than our problem in plasma physics, as may be seen from their descriptions in Section 2. Other plasma physics problems, however, such as those involving magnetically confined plasmas, would also be subject to this difficulty, which we perceive as the main one to be overcome in order that the potential of the MPP for particle simulation of rarefied media may be fully realised. The foregoing remarks relate to the first two questions raised in Section 1.

As regards the third question, we can already suggest several features that might be incorporated into a second-generation MPP, so as to improve its capability for particle simulation. There is a clear need for a larger random-access memory (RAM) associated with each processing element (PE), particularly if some future simulations are to be made in 3-D. At present, with only 1 Kbit of RAM per PE, we would be restricted to 2-D simulations with fewer than 10 particles per spatial cell, were it not that we are also using the staging memory for storing data on the positions and velocities of the particles; however, the use of the stager for this purpose entails a speed penalty. Another measure that might be taken to increase the computing speed is to introduce some degree of parallelism into the PE arithmetic; the present capability of varying the word length by 1-bit increments would probably have to be sacrificed, but this feature is less important in particle simulation than it is in image processing.

On the fourth and last question, as to the relative merits of the MPP architecture *versus* possible alternative architectures in the application to rarefied media simulation, we are now doubtful as to whether whether our present research program will yield a clear-cut answer. At the outset, we felt that the very simple way in which our square simulation domain can be mapped directly onto the square array of PE's made the MPP architecture a natural choice, but now we are less sure, in view of the acuteness of the load-balancing difficulty mentioned above. In any case, a final decision could not be taken until after some experimentation with other architectures such as the hypercube.

These considerations govern our plans for the future. In the short term, continuing the present research program through a second year, we shall endeavor to fully realise our immediate physics and computer science objectives by exploiting the possibilities of the MPP to the utmost. The physics objectives are to achieve realistic 2-D simulations of the chosen plasma and gravitational phenomena. The computer science objectives are to improve the simulation algorithms, notably by solving for the fields by direct numerical integration of Maxwell's equations rather than by transform methods. Moreover, in order to promote load balancing, we shall investigate the use of sort/merge routines to manage the particles, permitting the data concerning them to be stored in PE's other than those that correspond directly to their spatial positions in the simulation domain. Without going so far as to store these data in arbitrary locations, we feel that it may be helpful to allow them a certain latitude. This more flexible storage scheme should certainly help to reduce load unbalance resulting from random fluctuations of particle concentration in a statistically uniform medium, such as the plasma considered in our whistler instability simulation; it is unlikely, however, to be able to cope with unbalance resulting from large-scale gradients of particle concentration, as occur in our gravitational simulation.

In the long term, more powerful solutions to the problem of load balancing will have to be found. Probably the sole viable general solution is to divide up the simulation domain with a non-uniform grid, which is then mapped onto the processor array either directly, or with a certain degree of latitude as described above. If the large-scale distribution of particles is likely to change in the course of a simulation, then the grid may have to be adaptive as well. We hope to take part in these developments, which could be tried out on the MPP in its present form. Only when this problem has been adequately dealt with can a fair comparison be made with other architectures. We anticipate that the comparison is likely to favor the MPP architecture, and to provide strong motivation for the development of a second-generation MPP specifically for particle and fluid simulation.

ACKNOWLEDGEMENTS

The material above is based upon work financed by the National Science Foundation under Grant No. PHY-8506023, with additional funding provided by the NASA Ames University Consortium under Joint Research Interchange No. NCA2-144, and by the Stanford-Ames Institute for Space Research. The NASA Office of Space Science and Applications granted us free use of the MPP at NASA Goddard Space Flight Center (GSFC). We are most grateful to these organisations for their support. We also wish to express our gratitude to the NASA Space Physics Analysis Network (SPAN) pilot project for use of networking facilities, and to the staff of the Space Data and Computing Division at NASA/GSFC, particularly in the MPP User Support Office, for the many ways in which they have facilitated our work with the MPP.

We also wish to thank Dr. Bruce Smith, at NASA Ames Research Center, for advice and encouragement on the gravitational simulations, while, at Stanford University, our thanks are due to Prof. Oscar Buneman for similar guidance with the plasma simulations, and to Mr. Shepherd Shi and Mr. Igor Dayen for programming assistance.

REFERENCES

1. Buneman, O., C.W. Barnes, J.C. Green, and D.E. Nielsen, "Principles and capabilities of 3-D, EM particle simulations", *J. Comput. Phys.*, 1980, 38, pp. 1-34.

2. Bruno, J., "Final report on the feasibility of using the Massively Parallel Processor for large eddy simulations and other computational fluid dynamics applications", NASA Ames Research Center, Report No. RIACS–TR–84.2, 1984.

3. Bharuthram, R., and S. Baboolal, "Theory and simulation of the whistler instability driven by a velocity-space anisotropy", *S. Afr. J. Phys.*, 1985, 8, pp. 45-49.

4. Storey, L.R.O., and F. Lefeuvre, "Theory for the interpretation of a random electromagnetic field in space", in *Space Research XIV*, edited by M.J. Rycroft and R.D. Reasenberg, Akademie-Verlag, Berlin, 1974, pp. 381-386.

5. Storey, L.R.O., and F. Lefeuvre, "The analysis of 6-component measurements of a random electromagnetic wave in a magnetoplasma — I. The direct problem", *Geophys. J. R. Astr. Soc.*, 1979, pp. 255-269.

6. Lefeuvre, F., and C. Delannoy, "Analysis of a random electromagnetic wave field by a maximum entropy method", *Ann. Télécommun.*, 1979, 34, pp. 204-213.

7. Delannoy, C. and F. Lefeuvre, "MAXENTWDF — A computer program for the maximum entropy estimation of a wave distribution function", *Comput. Phys. Comm.*, 1986, 40, pp. 389-419.

8. Hockney, R.W. and J.W. Eastwood, *Computer Simulation using Particles*, McGraw-Hill, New York, 1981.

9. Dawson, J.M., "Particle simulation of plasmas", *Rev. Mod. Phys*, 1983, 55, pp. 403-448.

10. Birdsall, C.K. and H.B. Langdon, *Plasma Physics via Computer Simulation*, McGraw-Hill, 1985.

11. Okuda, H., and C.K. Birdsall, "Collisions in a plasma of finite size particles", *Phys. Fluids*, 1970, 13, pp. 2123-2134.

12. Ratliff, S.T., *Computer Simulations of the Toroidal Cusp Experiment*, University of California, Los Angeles, Ph.D. Thesis, 1983.

13. Morse, R.L. and C.W. Nielson, "Numerical simulation of the Weibel instability in one and two dimensions", *Phys. Fluids*, 1971, 14, pp. 830-840.

14. Boris, J.P., *Proceedings of the Fourth Conference on Numerical Simulation in Plasmas*, Naval Research Laboratory, Washington, D.C., 1970, pp. 3-67.

15. Buneman, O., "Fast numerical procedures for computer experiments on relativistic plasmas", in *Relativistic Plasmas*, edited by O. Buneman and W.B. Pardo, Benjamin, New York, 1968, pp. 205-219.

16. Hoshino, T. and K. Takenouchi, "Processing of the molecular dynamics model by the parallel computer PAX", *Comput. Phys. Comm.*, 1984, 31, pp. 287-296.

17. Johns, T.C. and A.H. Nelson, "Particle simulation of 3D galactic hydrodynamics on the ICL DAP", *Comput. Phys. Comm.*, 1985, 37, pp. 329-336.

18. Cooley, J.W. and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Math. Comp.*, 1965, 19, pp. 297-301.

N87-26537

A STUDY OF THE XY MODEL BY THE MONTE CARLO METHOD

by

Peter Suranyi and Paul Harten

ABSTRACT

The MPP is used to perform Monte Carlo simulations for the two dimensional XY model on lattices of sizes up to 128x128. A parallel random number generator was constructed, finite size effects were studied, and run times were compared with those on a CRAY X-MP supercomputer.

Keywords: XY model, Spin System, Monte Carlo, Random Number Generator

BACKGROUND

XY model

The XY planar model is a two dimensional latticework of interacting spins. Each spin is a vector of unit magnitude in the XY plane. The Hamiltonian of the system is

$$H = -J \Sigma S_i \cdot S_j = -J \Sigma \cos(\theta_i - \theta_j) \quad (1)$$

where J is the spin-spin coupling, S_i is the spin at site i, θ_i is the angle that S_i makes with an arbitrary axis, and the summation is over nearest neighbor pairs.

Unlike the Ising model which has a finite mean magnetization or long range order for temperatures below a certain critical temperature, the XY model has a different type of behavior in that there is no long range order for low temperatures. However, it does seem to have a critical temperature T_C where it goes through a phase transition.

Kosterlitz and Thouless show (ref.1) that there exists certain topological defects in the XY model which govern its unusual behavior at low temperatures. These defects are vortices of spin. Going around any closed path in the system the change in spin from site to site will add up to an integral multiple of 2π. This number is a constant of the system so that whenever a vortex is created.

Below the critical temperature, vortex-antivortex pairs are bound and stay close together. Above the critical temperature there is enough heat-energy in the system to break the bond and the vortices go off independently. This is the phase transition that the system goes through. It is under a class of what are known as defect mediated phase transitions.

Statistical mechanics

Using a heat bath algorithm, the probability of any state i of the system is given by the Boltzman factor

$$P(i) = \exp(-\beta H_i) = \exp(-H_i/kT) \quad (2)$$

where H_i is the Hamiltonian of the system, k is the Boltzman constant, and T is the temperature. States of the system are values of θ at each of the sites. It follows then that the partition function is given by

$$Z = \Sigma \exp(-\beta H_i) \quad (3)$$

where the summation is over all states of the system. The expectation value of any variable x is given by

N87-26537

$$\langle x\rangle = Z^{-1} \Sigma\ (x)\exp(-\beta H_i) \qquad (4)$$

where again, the summation is over all states of the system.

Monte Carlo

The Monte Carlo method approximates these expectation values by averaging the value of x over a random sampling of states near equilibrium, these being the states of highest probability. Since the values that the θ 's take on are continuous, the summation over all states is an integration from 0 to 2π over all θ. The random sampling near equilibrium then represents evaluations of this integral at a very sharp exponential peak.

The states near equilibrium are arrived at through a Markovian process where the transitional probability of going from state 1 to state 2, Q(1,2), follows the detailed balance condition

$$Q(1,2)P(1) = Q(2,1)P(2) \qquad (5)$$

where P(1) and P(2) are the probabilities of state 1 and 2 respectively.

The Metropolis method used sets Q(1,2) = 1 if the energy of state 2 is lower than the energy of state 1, and sets

$$Q(1,2) = P(2)/P(1) = \exp(-\beta\Delta E) \qquad (6)$$

otherwise. Here ΔE is the change in energy from state 1 to state 2. A little reflection will show that this indeed does fulfill the detailed balance condition. Not only does this method allow the system to approach the state of lowest energy (equilibrium) but it also allows the system to get out of any local minimum that it might find itself.

XY MODEL PROGRAM

System update

A 128x128 parallel array of real numbers is defined where the elements take on values from 0 to 2π. This is a lattice of spins and the numbers represent the angle θ of the spin at that site with respect to some axis. When any site is updated it depends only upon its four nearest neighbors so that for the purpose of updating there are actually two separate sub-lattices in a checkerboard pattern. One sub-lattice is updated at a time so as not to affect the updating of other sites. This conditional update is done, quite naturally, with a checkerboard mask.

For a site to be updated the θ values of its four nearest neighbors are brought into memory locations at that site using the ROTATE function. Along the sides of the lattice the ROTATE function brings in the θ values of the opposite sides, thus giving the system periodic boundry conditions. A trial spin is arrived at by randomly adding a value $\Delta\theta$, where $-\phi < \Delta\theta < \phi$, to the spin at that site. The value of ϕ is optimized for the fastest convergence. The difference in energy between the trial state and the original state is given by

$$\Delta E = H_t - H_o = -J\ \Sigma \cos(\theta_t - \theta_k) + J\ \Sigma \cos(\theta_o - \theta_k) \qquad (7)$$

where the t and the o subcripts are the trial spin and the original spin, and the summation index k is over the four nearest neighbors.

A random number R is then generated, if the energy of the new state is lower or if the random number $R < \exp(-\beta\Delta E)$ then the trial spin is kept, otherwise it is thrown away. This whole process is done for all sites on the sub-lattice in parallel. Following this, the complement sub-lattice is updated in the same manner and we have a new state

N87-26537

of the system near equilibrium.

Expectation values

The system is initially in a given state usually one of zero temperature where all of the spins are aligned. Then it is in effect brought into contact with a heat bath of temperature T. A warm-up period follows and the system is allowed to come to equilibrium at the temperature T. For the XY model, this warm-up takes on the order of 10,000 to 20,000 lattice updates. Sample values of x are taken every 100 updates to promote independence between values. Usually x will be a scalar property of the system so that the SUM function is used at this step.

The total number of samples taken are broken up into several equal divisions and x is averaged in each of these divisions. From these divisional averages both the total average of x and its standard deviation can be calculated.

Random number generator

The basic idea of the algorithm used to generate a plane of random numbers in parallel is as follows. Begin with two prime numbers a,p and set up an integer parallel array with a different power of a at every site where the powers range from 1 to 16384 (128*128). Multiplying every element by the constant a**16385 will always give a different power of a.

The next step is to take all numbers modulo p so that the randomness comes in as a**m mod p where m is some integer. Since

$$(ab) \bmod p = [(a \bmod p)(b \bmod p)] \bmod p, \qquad (8)$$

If we take modulo p at every step we will always have a**m mod p no matter how large m is.

Each time after multiplying by a**16385 mod p we have a new plane of random integers from 0 to p. To have a plane of random real numbers from 0 to 1, it is only necessary to divide by p.

This is an example of the parallel random number generator setup, which is done only once, and the generator itself. Here a=SEED, p=MAXRND, and IRND is the parallel array of integers:

```
PROCEDURE SETRND(VAR SEED,SEEDM:INTEGER
                    ;VAR IRND:ILAT);
  VAR I,J:INTEGER;
BEGIN
  SEEDM:=SEED;
  FOR I:=0 TO 127 DO
  BEGIN
    FOR J:=0 TO 127 DO
    BEGIN
      WHERE (ROW_INDEX=I) AND
          (COL_INDEX=J) DO IRND:=SEEDM;
      SEEDM:=SEEDM*SEED MOD MAXRND;
    END;
  END;
END;
```

ROW_INDEX and COL_INDEX must be declared as parameters in the PROGRAM line. SEEDM=a**16385 mod p

RAND is a parallel array of real numbers.

```
PROCEDURE GENRND(VAR SEEDM:INTEGER;
          VAR IRND:ILAT;VAR RAND:RLAT);
BEGIN
  IRND::13:=ROTATE(IRND::13,1,0);
  IRND:=IRND*SEEDM MOD MAXRND;
  RAND:=IRND/MAXRND;
END;
```

The bit plane ROTATE will be explained shortly.

RESULTS

Fourier components

For the purpose of finding some good seeds to use in this pseudo-random number generator, there are a few constraints to consider. It is necessary that p < sqrt(2**31-1) in order to insure that there is no erflow. Also, the fourier components of any random number generator must give results that are close to 0.

A program was written to look at the first 10 sine and cosine fourier components. From the results of this program it became clear that the multiplication by a**16385 mod p was not enough to insure randomness. But, the additional operation of rotating the thirteenth bit of every number in the random integer array one element north, did much to improve the randomness.

Two seeds found to give very good psuedo-random distributions are a=16307 and p=16309. Figure 1 shows the fourier components of the distribution generated from these seeds. The dashed lines on either side of zero represent one standard deviation assuming perfect randomness.

Given a perfectly random distribution and an infinite amount of numbers generated, the fourier components would be zero. Since finite amounts are being dealt with, there will be fluctuations away from zero even with perfectly random distributions. The points will do a random dance around zero. It is clear that these two seeds come very close to giving a random distribution.

More to the point is the result of using this psuedo-random number generator in simulating the XY model.

XY model simulation

The XY model simulation program was run at various values of β near the phase transition. Figure 2 shows the relation of the expectation value of nearest neighbor correlations, $<\cos(\theta_i-\theta_{i+1})>$, to β. The error bars are smaller than the points themselves. The lower curve is the result of a 128x128 lattice of spins and the upper curve is the result of a 16x16 lattice of spins.

The 16x16 lattice of spins was run for the purpose of comparison with some previous results done on a Cray X-MP. The results from the Cray are also shown in figure 2 and are sitting under the same points as generated by the MPP. This is exact agreement.

The distance between the curves of different sized lattices come from what are known as finite size effects. These are mostly due to the fact that in addition to direct spin-spin interaction there is also interaction coming around periodic boundries. In an infinite sized lattice, this would not happen.

The two curves come together at approximately $\beta = 1.15$. This is where the phase transition is taking place. Theoretical predictions (ref.2) give $\beta_C = 1.14$.

Cray-1 CPU time

From similar runs of an XY model simulator written for a Cray X-MP, very rough calculations show that it takes 1.36E-3 secs of CRAY-1 CPU time per lattice update. On the MPP, calculations show that it takes 1.86E-4 secs per lattice update. This is approximately seven times faster than the CRAY-1 time.

CONCLUSION

The MPP is a very useful tool in simulating the two dimensional XY model. Properties of the XY model can now be investigated which are realistically impossible to do anywhere else.

ACKNOWLEDGEMENTS

This work was supported in part by the US Department of Energy under Grant Number FG02-84ER40153

REFERENCES

1. J.M. Kosterlitz and D.J.Thouless,
J.Phys. C6(1973)1181;
J.M. Kosterlitz,J.Phys.C7(1974)1046.
2. D.C.Mattis,Phys.Lett. 104A(1984)357.

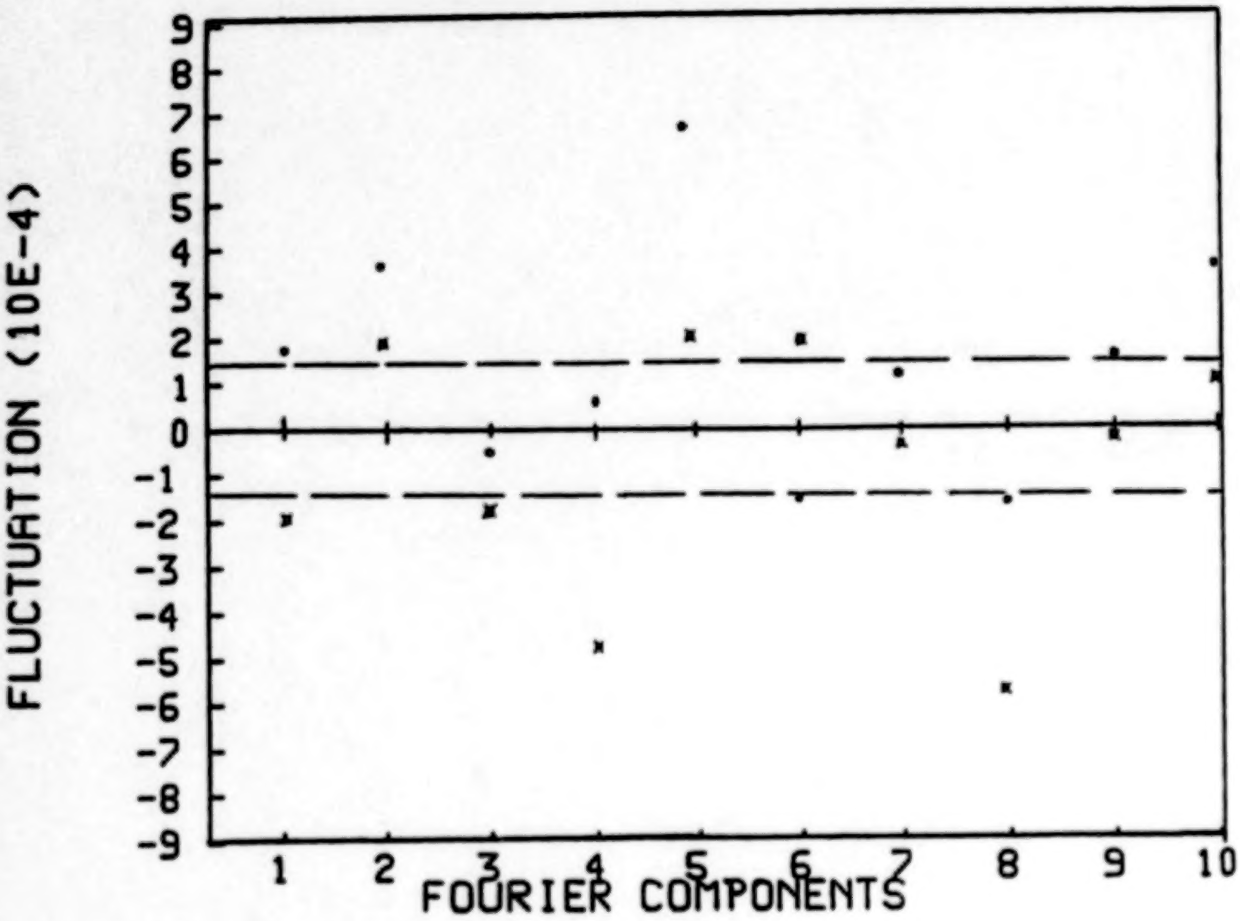


Figure 1. Random Fluctuations of the Fourier Components Away from Zero

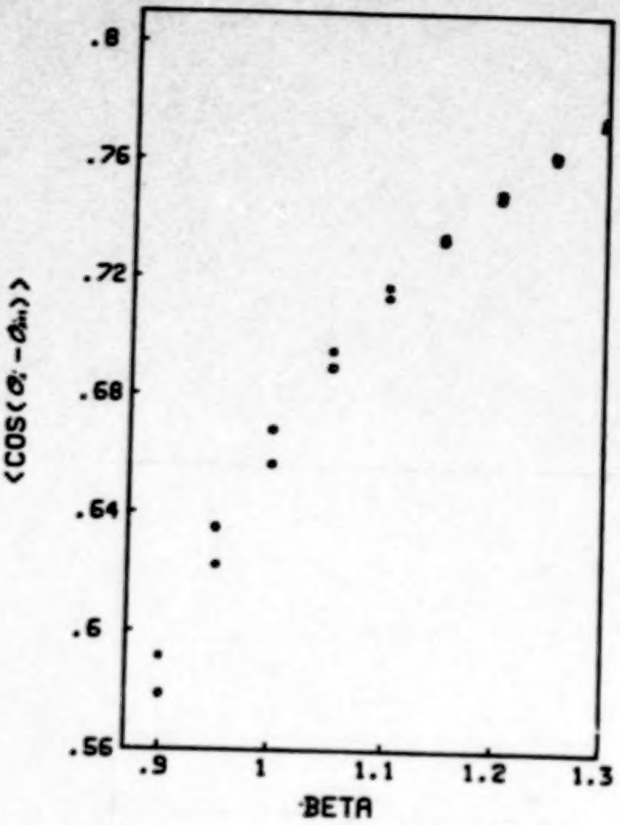


Figure 2. Nearest Neighbor Correlations vs β.

N87-26538

A Fast MPP Algorithm for Ising Spin Exchange Simulations

Francis Sullivan and Raymond D. Mountain
National Bureau of Standards
Gaithersburg, Maryland 20899

September 19, 1986

Abstract

Our main purpose is to describe a very efficient MPP algorithm for performing one important class of Ising spin simulations. Results and physical significance of MPP calculations using the method described here will be discussed elsewhere. However, we will make a few comments on the problem under study and report briefly on results so far. Ted Einstein has provided us with much guidance in interpreting our initial results and in suggesting calculations to perform.

1 Introduction

Ising spin simulations occur in many areas of physics and have attracted the attention of researchers since the earliest days of electronic computation (Ref. 4). They provide a very good example of how, from the point of view of algorithm design, two apparently disparate problem types can be attacked by almost the same techniques. Some classes of Ising spin calculations can require speeds far in excess of anything currently available.

The basic model is easily described. The spin variable $\sigma(i)$ is specified at the nodes of a uniform grid in two (or three) dimensions. At each grid site the spin can take on only the values +1 and −1. Spins are related to one another via the energy expression given by the Hamiltonian

$$H = -\sum_{\{i,j\}} \sigma(i)\sigma(j)$$

where $\{i,j\}$ ranges over all nearest neighbor pairs of sites. In an $\ell \times \ell$ square lattice, at site i we have the local energy expression

$$\begin{aligned} E(i) &= -\sigma(i) \times (\sigma(i+1) + \sigma(i-1) \\ &\quad + \sigma(i+\ell) + \sigma(i-\ell)). \end{aligned}$$

In most cases periodic boundary conditions are imposed, so that $i+1$ and $i+\ell$ are to be determined mod ℓ.

One wishes to compute various averages with respect to the probability $P(C)$ for a configuration of spins, C, to occur. The "classical" algorithm for using Monte Carlo methods to sample configuration space is due to Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (Ref. 4), (called the $M(RT)^2$ algorithm for short). It consists of a series of moves through configuration space, making use of the fact that for each C, $P(C) \propto \exp(-JE(C)/kT)$. Here $E(C)$ is the energy associated with configuration C, J is the coupling constant for the problem under study, T is the temperature and k is Boltzmann's constant. A site i is chosen at random and the change in energy $\Delta E(i)$ which would result in reversing the spin at that site is determined. Since only the site i and its four nearest neighbors are involved, it is easy to see that the change in energy is

$$\begin{aligned} \Delta E(i) &= (E'(i) - E(i)) \\ &\quad + (E'(i+1) - E(i+1)) \\ &\quad + (E'(i-1) - E(i-1)) \\ &\quad + (E'(i+\ell) - E(i+\ell)) \\ &\quad + (E'(i-\ell) - E(i-\ell)) \\ &= -4E(i) \end{aligned}$$

If $\Delta E(i) \leq 0$, then the move is "accepted" and the sign of $\sigma(i)$ is reversed. In case $\Delta E(i) > 0$, the move is accepted with probability $\exp(-J\Delta E/kT)$.

It can be shown that the $M(RT)^2$ algorithm defines a Markov process which samples the "correct" (Boltzmann) distribution of configuration of spins. However, for a large system several hundreds of thousands, or even millions of updates of each site must be performed in order to approach a single equilibrium configuration, and often averages over many such configurations are required. Most of the work is in generating the random numbers, since as many as 10^{12}

moves of sites may be required in a single simulation. We will delay discussion of this critical aspect for the moment, and concentrate on how to modify $M(RT)^2$ in order to get an algorithm well suited to the MPP. Later we explain why this algorithm is also well suited to vector machines, and we then describe one method for vectorizing the generation and testing of random numbers.

2 Algorithm

The MPP consists of a square array of 128x128 single-bit processors, having an SIMD architecture. Processors are interconnected via nearest neighbor bonds and periodic boundary conditions is one possible connection pattern for border processors. Because of its SIMD architecture, the MPP is extremely well adapted to image processing applications and, in fact, one point we hope to make is that, so far as algorithms are concerned, Ising spin simulations are a type of image processing problem.

One is tempted to think of associating processors with spin sites in a one-one manner. However, this does not make very good use of the machine, since in the $M(RT)^2$ algorithm, only one site is examined for each move. Instead of mapping sites to processors in the obvious way, we instead notice that the expressions for the energy associated with a site are similar in form to the central difference approximation used to solve the Poisson equation,

$$-\nabla^2 u = \rho.$$

In the Poisson case, a grid site and its four nearest neighbors are related through the finite difference expression

$$\begin{aligned} -u(i-\ell) - u(i-1) & \\ +4u(i) & \\ -u(i+1) - u(i+\ell) &= (\Delta x)(\Delta y) \bullet \rho(i). \end{aligned}$$

A common strategy used in implementing iterative methods for solving the Poisson equation is to use the "red-black ordering" depicted below:

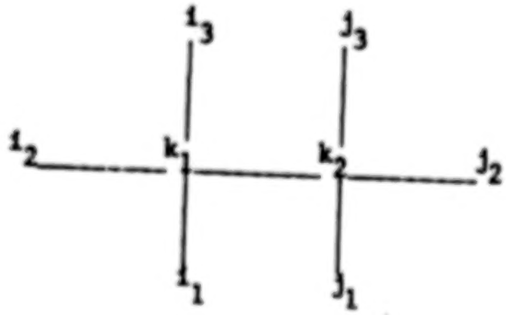
$$\begin{array}{cccccc} R & B & R & B & R & B \\ B & R & B & R & B & R \\ R & B & R & B & R & B \\ B & R & B & R & B & R \end{array}$$

Since no pair of red sites are nearest neighbors, all red sites can be updated "simultaneously". These values can then be used to update the black sites, and so-forth, alternating on each iteration between red and black sites.

The same idea can be applied to modify $M(RT)^2$. Spin flips can be attempted at all red sites or at all black sites. This is a different Markov process than $M(RT)^2$ but the same distribution is sampled. Of course it is the dynamical aspects which are of interest now, rather than merely the converged solution. This use of colors is part of the so-called multi-spin algorithm (Ref. 5). In order to implement multi-spin on the MPP, all sites of a single color can be associated with a single MPP plane of 128x128 processors.

The problem to which we have applied multi-spin is slightly more complicated than that of attempting to flip single sites (Glauber dynamics). Instead we want to try to exchange the spins of a pair of neighboring sites. These spin exchanges or Kawasaki dynamics calculations can be used to study phenomena such as growth of magnetic domains (Ref. 2).

Because spin exchanges are to be attempted, eight different sites are involved in each move. For example to interchange sites k_1 and k_2 we have all of the depicted bonds to consider. The change in energy is



the sum of changes over all eight sites and is given by the expression

$$\begin{aligned} \Delta E &= -2(\sigma(k_2) - \sigma(k_1)) \\ & \bullet \{\sigma(i_1) + \sigma(i_2) + \sigma(i_3) \\ & -\sigma(j_1) - \sigma(j_2) - \sigma(j_3)\}. \end{aligned}$$

In order to apply the multi-spin idea enough "colors" must be assigned so that no two sites of the same "color" are involved in the same move. This can be accomplished by partitioning sites into sixteen groups as is shown below:

15	16	13	14	15	16
11	12	9	10	11	12
7	8	5	6	7	8
3	4	1	2	3	4
		13	14	15	16

Each group is now associated with a single 128x128 MPP bit plane, so that a 512x512 simulation can be handled in a straight-forward way.

Notice now that the part of the expression for ΔE which is enclosed in brackets can take on only seven distinct values, namely $\{-6, -4, -2, 0, 2, 4, 6\}$. The MPP array consists of single-bit processors and Boolean operations can be done simultaneously in a single operation on all 2^{14} processors. The computation of ΔE can be reduced to Boolean operations by first associating the spin values $\sigma = \{+1, -1\}$ with Boolean values = $\{1, 0\}$ (which in effect makes a transcription to a lattice gas model), and next noticing that the value of the bracket part of ΔE corresponds to the sum of the 1 bits of the Boolean expression. For example:

Spin version:

$$\{1\ 1\ -1\ 1\ 1\ 1\}$$

Value = 4.

Boolean version:

$$\{1\ 1\ 0\ 1\ 1\ 1\}$$

Sum of bits = 5

The complete correspondence is as follows:

Spin Sum	Bit Sum	Representation
6	6	110
4	5	101
2	4	100
0	3	011
-2	2	010
-4	1	001
-6	0	000

The bit representation for the sum of bits can itself be determined by defining summation of Boolean expressions by Boolean operations. Assume that BITS(I), I = 1,2,...,6 is an array of bit planes each consisting of all 128x128 sites of a given color. The bit representation of the sum consists of three more bit planes B1, B2, and B3, initially all zero. Addition is performed bit by bit with the proper rules for carries. The correct representation for the final values is obtained by executing the following loop.

```
      DO  10  I=1,6
        B1 = B1 .OR. (BITS(I)
                .AND. B3 .AND. B2)
        B2 = (.NOT. (BITS(I)
              .AND. B3) .XOR. (.NOT. B2))
        B3 = (.NOT. BITS (I))
              .XOR. (.NOT. B3)
   10   CONTINUE
```

The sign of the sum of neighboring spins is determined by the value of the high order bit B1. This must be combined with the value of $-2(\sigma(k_2) - \sigma(k_1))$ to determine when a random number needs to be compared with $\exp(-J\Delta E/kT)$. Of course, the exponential also takes only finitely many values, so that it is easy to parallelise the comparison step. The generation of random numbers can be done simultaneously if a method which allows more than 2^{14} seeds is used. We have adapted a program due to Marsaglia and Kahaner (Ref. 3), but other methods are possible. In any case, 2^{14} pairs of sites are handled in a single step. The total number different types of nearest neighbor pairs ($< 1, 2 >, < 6, 7 >, < 9, 5 >$, etc.) is 32 , and one should not cycle through these types in a fixed pattern because this marching introduces false dynamics into the simulation.

3 Vector Machines

Essentially the same algorithm can be used on a vector machine such as the CYBER 205. The representations for ΔE can be computed very efficiently using bit vector operations. However, vector instructions are not really the same as parallel instructions, and so the 2^{14} random numbers which are required for each step take a lot of time to generate on a vector machine. The way to alleviate this is to reduce even the comparison with random numbers to vector operations on bit arrays. Assume, for convenience, that the value of $-2(\sigma(k_2) - \sigma(k_1))$ is -4, and let $\alpha = \exp(-8J/kT)$. As in (Ref. 1), we create bit arrays D1 D0 of length 2^{14} where D1D0 = 11 with probability α^3; D1D0 = 10 with probability $\alpha^2 - \alpha^3$, D1D0 = 01 with probability $\alpha - \alpha^2$ and D1D0 = 00 with probability $1 - \alpha$. This is easily done by generating 2^{14} random numbers and noting where they fall in the intervals I3 = $[0, \alpha^3)$, I2 = $[\alpha^3, \alpha^2)$, I1 = $[\alpha^2, \alpha)$ and I0 = $[\alpha, 1]$. The test can now be performed by computing the bits of (B1 B2 B3 + D1 D0) $\oplus$ 001. Here the operation $\oplus$ means addition modified so that the high-order bit is not "turned off" by a carry operation (e.g. 111 $\oplus$ 001 = 100). After this operation has been performed, exchanges can be made at all sites with high-order bit equal to 1.

A little reflection reveals that the above method performs exchanges with nearly the correct frequency, because the bit vectors D1D0 approximate the correct probability distribution. After D1D0 have been used for a step, the vectors must be permuted in some random fashion, and to insure that the false periodicities are not introduced, the D1D0 vector should occasionally be re-loaded by generating 2^{14} new random numbers. This, of course, is *much* more efficient than generating a full set of random numbers for each step.

4 Preliminary Results

The first program to implement the algorithm on the MPP was written in Parallel Pascal by Julie O'Connell. However, signigicant portions of it were later recoded in PEARL by Jim Abeles. The result is a code which performs better than 200 million tests of spin sites per second. Since our goal is to study the long time growth of domains, very long simulation times and averages over many different configurations are required. The present code makes this a practical possibility.

As has been mentioned, in the case of spin exchanges great care must be taken to avoid introducing any false periodicities into the results. This need for care is especially acute in our study because there is considerable controversy about the long time behavior of such models (Ref. 6). Preliminary runs on the MPP indicate that our algorithm is correct. We are now conducting more extensive tests of the reliability of our technique for using the random number generator.

References

1. Bhanot, G., Duke, D. and Salvador,R. 1986. A fast Algorithm for the CYBER 205 to Simulate the 3-D Ising Model.*J. Statistical Physics.* to appear.

2. Huse, D. A. 1986. Late Stages of Spinodal Decomposition: Corrections to Lifshitz-Slyozov Scaling and Monte Carlo Simulations. *preprint.*

3. Marsaglia, G. and Kahaner, D. Generation of Uniform and Normal Random Variables. *preprint.*

4. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. 1954. *J. Chem. Phys.* **21**: 1087.

5. Williams, G. O. and Kalos, M. H. 1984. A New Multispin Coding Algorithm for Monte Carlo Simulation of the Ising Model *J. Statistical Physics* **37**: 283-299.

6. Masenko, G. F., Valls, O. T. and Zhang, F. C. 1985. *Phys. Rev.* **B31**: 4453.

COMPUTER SCIENCE SESSION I

Session Chair: Prof. Michael A. McAnulty
University of Alabama in Birmingham

N87-26539

SIMULATION OF AN ARRAY-BASED NEURAL NET MODEL

John A. Barnden
Computer Science Department
Indiana University
Bloomington, Indiana

ABSTRACT

Research in cognitive science suggests that much of cognition involves the rapid manipulation of complex data structures. However, it is very unclear how this could be realized in neural networks or connectionist systems. A core question is: how could the interconnectivity of items in an abstract-level data structure be neurally encoded? My answer appeals mainly to positional relationships between activity patterns within neural arrays, rather than directly to neural connections in the traditional way. The new method was initially devised to account for abstract-symbolic data structures, but it also supports cognitively useful "spatial analogue", image-like representations. As the neural model is based on massive, uniform, parallel computations over 2D arrays, the MPP is a convenient tool for simulation work, although there are complications in using the machine to the fullest advantage. An MPP Pascal simulation program for a small pilot version of the model is running.

Keywords: neural networks, connectionism, representation theory, data structures, mental models, imagery, spatial analogues.

INTRODUCTION

Artificial Intelligence and Cognitive Psychology have provided the most advanced and detailed ideas about the nature of the information processing needed in cognition. Information processing as studied in those fields typically involves the rapid manipulation of complex, very-short-term symbolic data structures. The data structures may be representations of the world, semantic structures derived from natural-language utterances, descriptions of plans, goals, and their relationships, and so on. Such structures must be created, updated, analyzed, compared, modified, committed to memory, and retrieved from memory in rapid and complex ways. AI has been much concerned with the abstract nature of the data structures and manipulations, and with their implementation on conventional computers.

I therefore adopt the working hypothesis that much of human cognition requires *rapid computational processes involving complex, temporary structures* that hold information. These structures do not, however, have to be very similar to the particular data structures proposed in current AI and cognitive science.

Now, in recent years there has been great interest in the question of how "brain hardware" operates so as to support cognitive functions. The usual assumption is that the proper hardware level at which to study cognitive functions is that of the dynamics of neural *networks*, in which individual neurons perform only quite low-level functions and communicate with each other by sending simple signals along fibres. I take this assumption as a second working hypothesis, even though I am intrigued by suggestions that other effects may be important; that, for instance, individual neurons may be capable of performing major pieces of computation in their own right.

The two working hypotheses lead to the central question addressed by the project:

> CENTRAL QUESTION
> How can rapid processing of complex, temporary information structures, of some sort or other, be accomplished by "standard" neural net circuitry?

The emphasis on *short-term* structures and processing is to be noted. The project is not at present concerned with matters of long-term memory or learning. A late stage of the project will look at such matters.

Limitations of Prevailing Ideas

Most of the research on the cognitive capabilities of neural networks has centered on visual pattern recognition, lexical processes in language understanding, the "associative" retrieval of information from long-term memory, and restricted forms of learning. But for us to progress beyond these particular specialized

PRECEDING PAGE BLANK NOT FILMED

functions, there must be a more thorough and general examination of how neural networks can represent and manipulate information. As is widely admitted, it is very unclear how we could answer the Central Question by applying the main current ideas about neural networks.

Most work on neural nets in AI and cognitive science is within the so-called "connectionist" paradigm [Refs. 4, 6, 8, 11, 14]. Some connectionists have recently begun to look at aspects of the central question [e.g. Ref. 13]. However, the work is in its infancy and has barely begun to address the real issues. There is reason to believe [Ref. 1] that it will be difficult to deal with these issues without adding a great deal of extra complication to existing connectionist models.

My own neural net model is a more thoroughgoing attempt to confront the issues than has previously been made. In this confrontation, I have found it beneficial to introduce some novel ideas about neural representation. The theory is thus highly atypical as a connectionist theory if one defines connectionism by appeal to existing research, although it is still connectionist in that it relies on networks of processing units that are like simplified neurons and send only simple messages to each other. The model is conjectural, though respectably in tune with neurophysiological evidence [Ref. 2]. An important feature is that it serves as a brake on uncritical acceptance of the (similarly conjectural) existing views of neural-net functioning.

The Central Representational Tenets

Here I specify the basic, general representational principles underlying the project. They could conceivably by instantiated in detail in numerous different ways. One specific instantiation that I am studying and simulating will be outlined below. However, the general principles would still be of interest even if this instantiation were abandoned.

PRINCIPLE 1

> The neural realization (implementation) of a short term data structure is composed of localized patterns of neural activity. These localized patterns are *positioned occurrences*, in some neural medium, of patterns of neural activity that are location-unspecific in that they can be instantiated at any position within that medium.

Analogy: A piece of natural language text is a data structure (in a paper/ink medium) whose parts are localized occurrences of location-unspecific visual patterns (words).

PRINCIPLE 2

> One important way in which those neural activity-pattern occurrences are "glued" together so as to form structures is to be in appropriate *relative positions* in the neural medium.

Analogy: In a piece of text the word occurrences are structured by being put into appropriate relative positions.

PRINCIPLE 3

> Activity-pattern occurrences that act as parts of an implemented data structure can also be associated with each other by being similar in some particular sense.

Analogy: a textual label used in a diagram and in a legend attached to the diagram serves to associate a part of the legend with a part of the diagram.

There is a fourth principle that I state later.

Neural Arrays

Principles 1 and 2 mentioned neural-net media in which patterns of neural activity can be given instantiations in particular positions. Thus, I am assuming that the media are such that one can make sense of the notions of "instantiation", "position" and "relative position" within a medium. To this end I have adopted the following simple assumption in my model: each medium is a *2D square array of small neural circuits*. All the arrays are of the same size, and all the small neural circuits (the array elements) are similar to each other. Each array element supports at any given time a particular pattern of neural activity, and, since the array elements are isomorphic circuits, the supported patterns can be viewed as positioned instantiations of location-unspecific patterns.

A short-term data structure, then, consists at the hardware level of configurations of pattern instantiations in one or more of the neural arrays. To continue the text or diagram analogies mentioned above, an array can be likened to a piece of paper on which words can be put in particular places.

In theory the arrayness of my neural media could be purely abstract, with no implication that they are laid out in space as geometric arrays. Nevertheless, I do in fact assume that they are so laid out, at least approximately. This is convenient because the processing applied to the media requires adjacent array elements to communicate. Although there is some degree of broadcasting within arrays, it is overshadowed by the neighbor-neighbor interactions.

The 2D geometric arrayness of the media fits in well with current ideas about the functional structure of cortex. I say more on this later.

Spatial-Analogue Representation

Now that we have proposed an array-based computational architecture, it is a short step to the following proposal: that bodies of information that can abstractly be cast as arrays could in some cases be implemented in the architecture by adopting a natural mapping from the abstract arrays down to our neural arrays. (Cf. the standard implementation of a 1D abstract array by sequential allocation in a conventional computer memory, viewed as a 1D array of cells.) The point of this sort of "natural" implementation of abstract arrays is that it can render certain sorts of operation on the abstract information more efficient or more convenient than they would otherwise be.

There is one specific sub-proposal that much of my project is concerned with: that *2D "spatial" arrays could be naturally mapped onto the neural arrays.* By a " 2D spatial" array I mean an abstract array in which element (i,j) is meant to correspond to position (or subregion) (i,j) in some 2D projection of space. Thus, in a neural array that is being used to support a spatial array, the element (i,j) would correspond to and hold information about position (i,j) of the 2D projection of space.

Of course, having arrived at this point we observe that a neural array could be used to naturally map an *abstract* 2D space as well as a geometric one. Altogether, we are led to the following:

PRINCIPLE 4

> Relative position of activity-pattern occurrences in a neural array may (from time to time) serve to encode relative position in (geometric or abstract) spaces.

This principle as stated is very general. One important possibility that it allows is for spatial-analogue representation to be mixed with other sorts of representation in the very same neural array. Such hybrid types of representation are a major focus of my project.

I view the range of possibilities allowed by Principle 4 as constituting a neurally-explicated capability for a form of *imagery*. I use the term "imagery" to link the work to studies of imagery by psychologists and philosophers [Refs. 3, 7, 10]. Imagery is held to confer processing advantages in various sorts of mundane cognition, including some types of problem solving, although there has been much controversy over the various claims made, and indeed over what imagery is in the first place. On the other hand, although I find it convenient to use the term "imagery" in talking about my model, I emphasize that I do not imply that human beings have any conscious access to the "images" I postulate. The images are simply a certain class of data structures, processed in a certain way.

THE SPECIFIC MODEL

I now sketch a particular, detailed model that instantiates the data structuring principles presented in the previous section. A longer discussion is provided elsewhere [Ref. 2].

The role that this particular instantiation of the principles plays in the research should be carefully understood. I view it as just one of the many conceivable detailed models that conform to the general principles. It is this class of models that is my real interest. Thus, it is possible that the present detailed model may later be overthrown in favour of another. In any case, the present model contains several deliberate over-simplifications.

The instantiation takes the form of an information-processing model specified at a level slightly above that of detailed circuitry. That is, the model is composed of building blocks that can clearly be given a straightforward implementation in terms of idealized neurons that typically act like logic gates.

The model is based on a set of neural arrays of the sort described above. I henceforth call them *configuration matrices* (CMs). A short-term data structure is set up by putting appropriate CM elements into appropriate activity states (and letting the others be in a "null" or "resting" state). The elements used in a data structure need not all be in the same CM.

Structure per se consists partly in the relative positioning within individual CMs of the CM-element states. (See Principle 2.) This form of structure is backed up by a version of Principle 3: if two CM elements are in states that are similar in a certain way then they are taken to be identified with each other; that is, certain processing mechanisms treat them as if they were the same element.

A CM is laid out on the cortical surface as a geometrical 2D array, at least approximately. Each of the CM elements composing a CM is connected to its eight neighbors, to enable certain basic processing operations. The fact that the theory proposes several CMs rather than just one larger one reflects a concern with physiological plausibility, but there is no space here to go into this matter.

The total set of configuration matrices is divided into a small set of "principal CMs" and a possibly large set of "storage CMs". The activity configurations (i.e. implemented data structures) in the principal CMs can be acted on by condition-action rules. These rules are implemented in neural circuitry that is for the most part outside the CMs themselves but is also dependent on connections between CM elements. Rules do such things as responding to, analyzing and modifying the data structures in principal CMs, copying them into and out of storage CMs, organizing transfers of information between CMs and long-term memory, and sending signals to mechanisms external to the CM model itself.

I emphasize that *the data structures in CMs are short-term.* They are constantly being modified, in the service of cognitive tasks such as language processing, commonsense problem-solving, planning and so on. The nature of long-term memory in the theory will only be investigated at a late stage in the project. (I certainly do not assume that a long-term encoding of a CM activity-configuration is itself a pattern of neural *activity*.)

The state of a CM element at any moment t is an ordered pair (B_t, H_t). The component B_t is one out of a finite set of values called "basic symbols". This set of possibilities is the same for every CM element. Any number of CM elements can simultaneously have the same B_t value.

Most basic symbols will, at least in the initial stages of the project, be neural activity patterns that are internal "names" for entities such as particular people, classes of objects (e.g. the class of all tables), classes of situations or events, etc. When a CM element holds such a basic symbol then we regard the element as (*currently*) representing the entity named by the symbol.

The H_t component of a CM element's state is an ordered tuple of ON/OFF flag values. For intuitive convenience, each tuple position is identified by the name of a color. When, say, the "red" position contains the value ON we say that the CM element is (*currently*) "highlighted in red", or just "red".

Data structures are created by placing particular basic symbols in various CM elements and/or by putting CM elements into particular highlighting states. The absolute positions of basic symbols and highlighting states in a CM are irrelevant in the case of data structures with no spatial-analogue aspect — only the relative postioning is significant. For the purposes of illustration, suppose there is a basic symbol that is the internal neural "name" for a particular person Mary. Let us refer to this basic symbol as MARY. Similarly, suppose there is a basic symbol JOHN for a particular person John, and a basic symbol LOVE that is the internal name for the class of possible loving situations. Then the statement that John loves Mary can be encoded by means of the CM sub-configuration depicted in Figure 1.

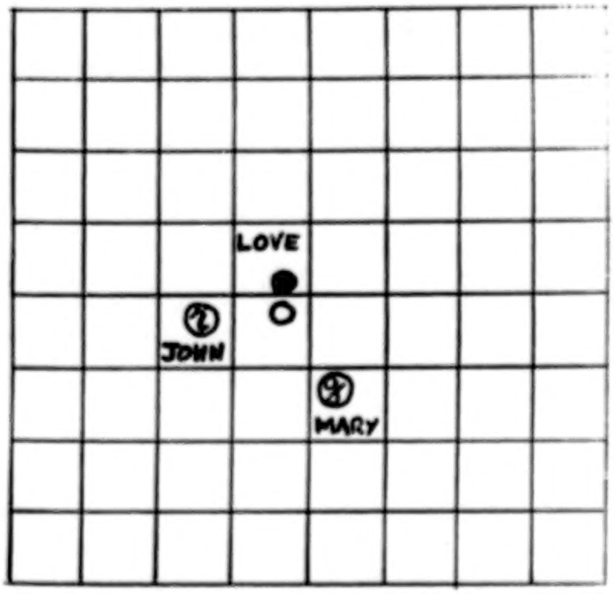


Figure 1: John loves Mary.

The figure is to be understood as follows. It illustrates an 8 by 8 region within a CM. Squares illustrate CM cells. (The illustration is schematic: CM cells are not meant to be geometrically square regions of cortex.) Capitalized words inside the squares illustrate basic symbols, the black disc illustrates highlighting in black, the empty circle illustrates highlighting in white, and 'r' and 'g' in circles illustrate red and green highlighting respectively. Squares with no basic symbol shown illustrate cells containing a special NULL basic symbol that does not represent anything. Basic symbols JOHN and MARY represent the people John and Mary, and basic symbol LOVE represents the "love" situation-class. The highlighting serves to distinguish the roles of adjacent CM cells with respect to each other. White and black highlighting are confined to the special use illustrated here: a white CM cell represents a member of the class represented by the adjacent black CM cell. Therefore the white cell in the Figure represents an individual love situation, and is the "head" cell of the statement. In a love statement the agent and object cells (those containing JOHN and MARY here) must be highlighted in red and green respectively to indicate their particular roles.

The most important thing to notice about the example is the use of *adjacency* between CM elements as the "glue" sticking components of a data structure together. The particular relative positioning of CM states is unimportant except in so far as the appropriate adjacency relationships are achieved. Note that adjacency includes diagonal adjacency.

More complex data structures are built up in much the same way, as shown elsewhere [Ref. 2]. However, the representation method as so far presented suffers from an overcrowding problem. Most obviously, the method as it stands cannot handle an n-ary predication for n > 7, rather than a binary predication (using "loves", say). The overcrowding problem is avoided by the use of "unassigned symbols", a special type of basic symbol. An unassigned symbol does not permanently name anything. Rather, a CM element containing such a symbol can temporarily name something by virtue of that element's current role in a data structure. The crucial point is that if any two CM elements contain the same symbol, then they are in a sense identified with each other. Roughly this means that the two elements both currently represent the same thing. Therefore, data structures can be split up into sub-configurations.

The technique is illustrated in Figure 2, showing how the information that *John loves Mary or Mary loves John* could be encoded. In this figure the letters 'k' and 'l' indicate unassigned symbols, temporarily naming the hypothetical love situations. The OR subconfiguration uses those unassigned symbols, thereby referring to the same love situations. The 'p' in circles indicates purple highlighting.

The sharing of symbols by CM elements constitutes the model's instantiation of Principle 3 above. Two elements are in a "similar" state just when they contain the same basic symbol, and the "association" involving them consists in their being identified with each other. It is only fair to mention that similarity association, although perhaps conceptually elegant as a representational technique, introduces some extra complexity and inefficiency into the processing mechanisms.

Logical quantification can be handled straightforwardly, by means of activity configurations that are analogous to quantified formulae in first-order logic. However, I am not happy about proposing brain data structures that are similar in this way to logic formulae. I am interested in the possibility of including more ad hoc or naive quantificational machinery. I make one suggestion below.

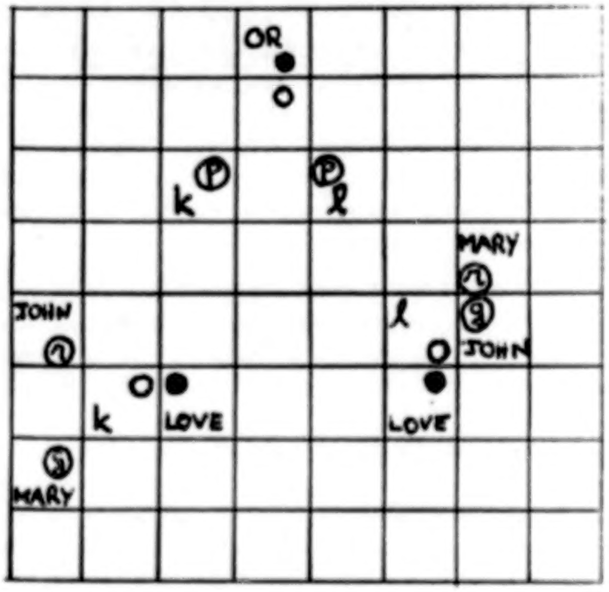


Figure 2: Structure splitting.

An Important Caveat

My interest is more on how the representational Principles enunciated above can be used to effect data structuring than in what the content of the data structures is. Thus, data structures and rules of the type illustrated in this section are merely being used for intuitive convenience. I am *not* committed to the data structures and rules being at just this conceptual level in the brain. For instance, it may be that statements would be broken down into lower-level conceptual primitives. Nor am I committed to the (abstract-symbolic) data structures being closely analogous to logical formulae, as they are in the present model.

Some Parameters

The project takes CMs to be of size 32 by 32. This size fits well with my conjecture that each CM element is implemented in the brain as a "minicolumn" of cortex [Ref. 9]. Minicolumns are claimed by some neurophysiologists to be basic functional units of cortex. There are about 1000 in each of the larger columns of cortex (about 1mm wide) that also seem to be important functional components. Thus, a CM of size 32 by 32 fits nicely into a large column. A minicolumn contains about a hundred neurons, which is enough for my purposes, especially since a single neuron is itself capable of complex switching behavior. (Real neurons are much more complex than single logic gates.)

If there are basic symbols of the sort used in examples above, then there must be several thousand of them (and an accordingly large number of location matrices — see below — for each principal CM). The simulations in the project will use an artificially restricted set of basic symbols, numbering no more than a hundred or two.

I conjecture that there are only a handful of principal CMs in the brain, and the simulations will certainly not use more than a dozen or so. No a priori limit to the number of storage CMs will be imposed.

Spatial and Diagrammatic Imagery

Line-of-Sight Tasks — In accordance with the spatial-analogue Principle (number 4), I allow relative position of CM elements to be used temporarily to represent relative positions in geometric spaces or in abstract spaces. In particular, if both "spatial" axes in a CM are taken as analogues of geometric axes, then a CM configuration acts as a sort of "spatial image".

Figure 3 depicts an example of a spatial image. Such a data structure might be used in dealing with a task mentioned by Sloman [Ref. 12] — determining whether some adversary A can see desired goal object G, there being an obstacle that might block A's line of vision. In the Figure, the 'G' and 'A' indicate CM cells representing the goal object and the adversary. The cells could contain basic symbols representing those entities, or could be suitably qualified by abstract-symbolic structures in which the cells are embedded (though none are shown in the Figure). The large blob illustrates an irregularly-shaped group of orange-highlighted CM cells depicting the rough shape of the obstacle. The crosses indicate a group of gold-highlighted cells that are approximately in a straight line. This group depicts the line of sight. Notice the use of highlighting to differentiate the identities of various entities. All the cells illustrated are highlighted in blue as well. Certain processing rules take the relative positions of blue cells (and only blue cells) to indicate the relative spatial positions of the represented objects. The obstructing of the line of sight can be detected by a rule that responds to CM elements that are highlighted in both gold and orange.

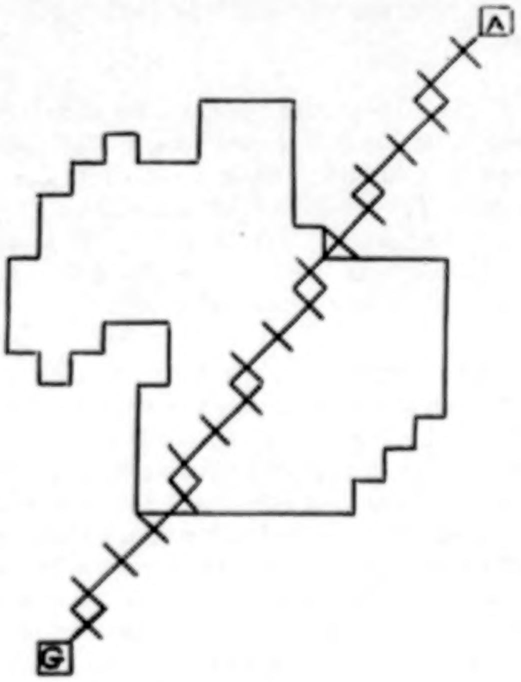


Figure 3: Line of sight.

The spatial problem-solving achievable with such images is obviously of an approximate nature, but is nevertheless heuristically useful. Elsewhere [Ref. 2] I give the details of the processing of such an image. Notice that the gold line represents a non-physical object, but itself has the same spatial quality as the blob representing the physical obstacle. This phenomenon is part of the "diagrammatic" quality of the system's imagery capability. Observe also the very important point that the image could include abstract statements about the entities involved. E.g. there brain model than the standard logical forms of quantification are.

RULES

We now turn to the way the rule mechanisms work. Consider an inference rule that could be stated in English as:

`if a man loves a woman then the man is hungry`

and assume that this rule is realized as a body R of neural circuitry (mostly external to the CMs). Suppose there is a principal CM holding a data structure stating that man M loves a woman W. (Recall that a principal CM is a CM whose contained data structures can be manipulated by the rule mechanisms.) To put it briefly, R detects the data structure by means of suitable input taken direct from the CM and indirectly from it via various "location matrices" (LMs). The detection is achieved by circuitry that operates in parallel uniformly over the whole of each LM or CM concerned. R then sends signals to LMs and the CM that cause a piece of data structure stating the man is hungry to be inserted into the CM. Once again, the signalling operates in parallel over whole arrays. For instance, when signalling to the CM, R sends the same signal simultaneously to all CM elements. These then respond differently according largely to their state of highlighting and the highlighting states of their immediate neighbors.

In the following I discuss only the detection phase of rule firing. The crucial feature of this phase is the use of LMs. Consider a single principal CM. For each ordinary basic symbol B there is a location matrix LM_B associated with the CM. This is a 2D neural array isomorphic to the CM, although the LM elements themselves are not isomorphic to CM elements. For now we can assume that the state of an LM element is just a single flag value. We say that the LM element is on when the value of this flag is ON. The purpose of LM_B is to indicate the positions of those CM elements that contain the symbol B. Specifically, a element in LM_B is ON when and only when the corresponding CM element contains B.

If B is a symbol naming a *class* of entities, then there is a *member location matrix* MLM_B as well as the matrix LM_B. The task of MLM_B is to indicate the positions of *members* of the class. For instance, the basic symbol LOVE used earlier names the *class* of conceivable loving situations. Elements that are on in LM_{LOVE} indicate the positions in the CM of occurrences of the LOVE symbol itself. On the other hand, elements that are on in MLM_{LOVE} indicate the positions of elements in the CM which explicitly represent *individual* loving situations currently. The most important case of such a element is a "love-proposition head element" — that is, a white-highlighted element which is next to a black-highlighted element containing LOVE (cf. previous Figures). Another case is when a CM element contains an unassigned symbol that is also sitting in a love-proposition head element, for then the two elements represent the same loving situation.

Similarly, the task of MLM_{MEN} is to indicate the positions of CM elements that represent individual men. This works analogously to the LOVE case, since a white element next to a black element containing MEN represents some man. Of course, a CM element might contain the basic symbol JOHN, where John is a man. Thus, whenever an element in LM_{JOHN} is turned ON the corresponding element in MLM_{MEN} is also turned on (if it is not already on). Similarly, whenever a element in MLM_{MEN} is turned on the corresponding element in $MLM_{PERSONS}$ is also turned on.

The example rule mentioned above requires a further location matrix, to indicate the positions of CM elements representing man-loves-woman situations. This matrix is described elsewhere [Ref. 2]. Notice that it is needed *only* because some rule needs to detect man-loves-woman situations. *No location matrix is necessary unless it forms part of the machinery for some rule.*

Similarity-association or symbol-sharing is given computational flesh by LMs. The basic idea is that whenever element (x,y) of a given LM is ON and the corresponding element of the CM contains a non-null symbol B, we want the system to turn ON every other element (x',y') (in that LM) such that element (x',y') in the CM also contains B. This effect is just what is required for symbol-sharing to be taken as CM-element identification. In fact, to facilitate the process I take each LM element to contain the basic symbol that is in the corresponding CM element. The required spread of ONness in an LM is achieved by each ON element in the LM broadcasting its symbol to all other elements in the LM, and every element then turning ON if its own symbol is the same as the one broadcast.

Without special assumptions this broadcasting must be done separately for every different symbol represented in the LM, but if in fact the symbols are nearly-orthogonal vectors of neural activity then it might be a statement to the effect that the adversary is afraid of "us" (the agent doing the problem solving). This ability to mix spatial and non-spatial data structures is another aspect of the diagrammatic quality of the system.

The rules that operate on the spatial image in the line-of-sight task rely predominantly on primitive signalling mechanisms that are much the same as those required in any case by non-spatial-analogue, abstract-symbolic processing. This point is an important aspect of the fact that the theory puts spatial imagery and abstract-symbolic processing into a unified framework.

Johnson-Laird's Mental Models — Johnson-Laird has devoted much attention to human reasoning about physical arrangements of objects [Ref. 5]. In a simple case, a subject is told that the fork is to the left of the spoon and that the knife is to the left of the fork, and the subject must infer or verify that the knife is to the left of the spoon. Johnson-Laird maintains that the subject constructs not only some internal propositional representations of the given pieces of information (these representations being of a form close to the linguistic statements themselves) but also a "`spatial` mental model" of the situation. This model consists of some sort of internal map-like representation of space: tokens standing for the various objects in the task are placed in particular positions in the map, the positions being consistent with what the propositional representations say. The desired relationships can then be read off the map directly.

The present project will provide a rather direct neural implementation of Johnson-Laird's models. That is, in a CM being used as a space map, elements can be made to represent particular objects by means of the data structuring techniques alluded to in the previous section. These elements would be in relative positions in the CM that correspond to the relative positions of the object relationships as specified in the task statement. See Figure 4. The three white cells represent a knife, a fork and a spoon. These cells are also blue, to indicate that their relative positions have spatial significance. For illustrative convenience, leftness/rightness in the CM illustration in the Figure corresponds to leftness/rightness in physical space.

In fact, an important feature of Johnson-Laird's total theory is that mental models are constructed from propositional (abstract-symbolic) representations that state the object relationships in the case at hand. His propositional representations would have an analogue in my system, in the form of abstract-symbolic data structures in various CMs. "A is to the left of B" can of course be represented in the system in a way similar to the way "John loves Mary" is represented. These data structures will be used as a basis for constructing the diagrammatic spatial images that display the relative positions. The construction will itself be done by rules in the rules component of the model.

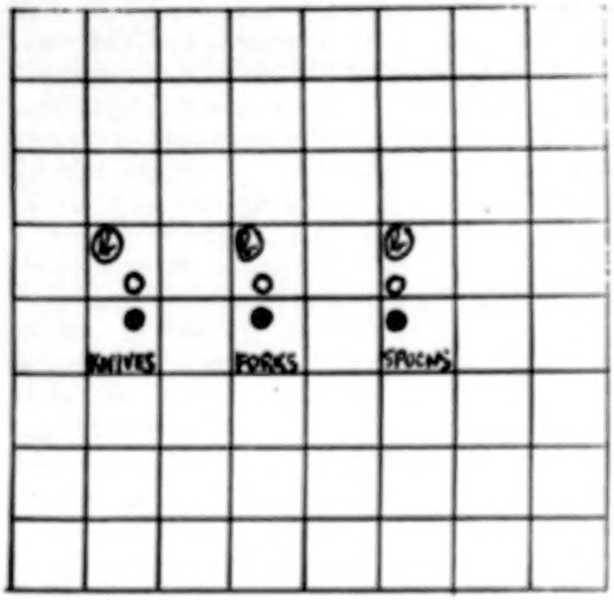


Figure 4: An implemented mental model.

For various reasons, Johnson-Laird suggests that some `spatial` mental models may contain simple symbolic devices of a propositional nature as well as the devices already illustrated. The CM theory is well set up to handle this sort of mixture, since it is already geared to hybrid, diagrammatic imagery as well as simple spatial imagery and purely propositional data structures.

The project will give precise computational substance to another major aspect of Johnson-Laird's ideas. This is the issue of syllogistic reasoning. Johnson-Laird proposes the use of another class of mental models to cope with such reasoning. This class is easily encompassed in my model. The syllogistic mental models involve certain restricted forms of quantification that are arguably more plausible for a turns out that the treatment of all the different symbols can proceed in parallel. I am investigating means for achieving this near-orthogonality.

Interpreted Rules

There is a major disadvantage to the hardware implementation of rules of the sort assumed above. The rules themselves are not data structures that can be inspected and manipulated (in the service, say, of rule learning and modification). A major task of the project will be to investigate the possibility of rules in the form of data structures. That is, to continue the example used earlier, the rule "if a man loves a woman then the man is hungry" would be a data structure of some form in a CM. The hardware rule mechanism would then consist partially or wholly of rules that act together as an *interpreter* of the data structure rules.

The ideas on interpreted rules are less fully developed than are other aspects of the model. A detailed syntax of data structure rules remains to be fully defined, and the nature of the pattern matching process required for rule enabling is not yet completely worked out. I anticipate that this process will operate largely by the flow of simple signals among CM elements.

MPP SIMULATION

An MPP Pascal simulation program for a small pilot version of the model is running. In this version there is just one CM, 20 basic symbols (the location-unspecific neural activity patterns that can be instantiated in CM elements), 5 highlighting colours, 2 very simple rules, and 22 LMs (the location matrices subserving the rules). Later simulations of the model will involve a hundred or two basic symbols, a dozen or so highlighting colours, a few principal CMs, dozens of storage CMs, dozens of rules per principal CM, and several hundred LMs per principal CM.

The current level of simulation is above that of detailed neural circuitry, so that for instance a basic symbol is simply simulated at present by a integer value. The highlighting flags at each CM element are simulated by boolean values. The ON/OFF state of an LM element is also simulated by a boolean value.

Accordingly, the CM is simulated by (1) a parallel array of integers (in a subrange) and (2) for each highlighting colour, a parallel array of booleans. Each LM is simulated by a parallel array of booleans. All these arrays are simultaneously present in the ARU. Since in the model each CM and LM is of size only 32 by 32, only parallel subscripts in the range 0..31 have significance currently.

After the initial state of the CM has been set up from input data, the simulation proceeds cyclically. Each cycle consists of (a) redefining the state of every LM, (b) firing one rule (if any can be fired at all), thus usually altering the state of the CM. The cycling stops when either no rule can fire or an input-specified number of cycles has been performed.

The neural model itself relies mostly on uniform, local, parallel operations over any given matrix (i.e. a given operation involves all elements of a matrix simultaneously, the processing at the elements varying only according to their own current states and their neighbors' current states). Therefore, most of the programming of the LM-state updates and rule-firing is straightforward, and makes good use of the ARU's SIMD nature and of MPP Pascal's parallel-array features. A certain amount of array shifting is needed in those cases when the processing at a matrix element brings in the neighboring elements.

In later model versions, there will be several CMs, and more than one will be "principal" (i.e. associated with rule machinery). Since there will be significantly many periods during the model's processing when various different principal CMs (and attendant LMs) are being similarly processed, and since it is unlikely that any model version will contain more than 16 principal CMs, it will be convenient to simulate the principal CMs by using different 32 by 32 portions of the ARU (with the LMs attached to a given CM using the same ARU portion as that CM). Thus various principal CMs will be able to be operated on simultaneously. However, the arrangement will complicate the programming of the frequently needed copying of the contents of one principal CM into another principal CM.

Storage CMs, on the other hand, can be conveniently stacked vertically beneath principal CMs, as there would be little benefit in arranging them horizontally across the CM, and they do not have any associated LMs.

LM Paging

In later versions of the model, each LM element will have a few flags in addition to the single on/off state mentioned earlier. These flags and the state will be simulated by booleans. Now, in the model, each LM element will also contain a copy of the basic symbol in the corresponding element of the CM to which that LM is associated. However, it turns out that my current hypotheses about the use of these symbol copies are such that there will be no need to carry the duplication over into the simulation itself. Thus, it will be adequate to go on implementing each LM as a small set of boolean parallel arrays. Nevertheless, since large versions of the model will contain several hundred LMs for each principal CM, the memory capacity of the PEs in the ARU will be too small to hold all the LMs. A scheme will be needed for paging LMs to and from the staging memory.

A Snag

To oversimplify a little, I would have liked to use the following piece of program to do some basic LM updating. (Type `symbols` is a subrange of `LMids`, which is itself a subrange of integer.)

```
(* CM_symbols:  parallel array[0..127,0..127]
                              of symbols;
   LM: parallel array[LMids,0..127,0..127]
                of boolean;
*)

where   CM_symbols <> null_symbol
   do   LM[CM_symbols]:= true
```

That is, the desired action was:

for each i,j where `CM_symbols[i,j] <> null_symbol`,

set `LM[CM_symbols[i,j], i,j]` to true.

Unfortunately, albeit understandably, neither the hardware nor MPP Pascal supports this "indirect addressing" at PEs. Therefore, I have had to resort to a loop that cycles through the (non-null) symbols, using a control variable `s`, doing

```
LM[s]:= (CM_symbols = s)
```

at each stage. This loss of parallelism will not have a drastic effect on my own simulation work, but the phenomenon is of some general interest. Goodyear Aerospace's Dr. Ken Batcher, designer of the MPP, suggested indirect addressing at PEs as an important addition in future versions of the machine, in his invited speech during the Symposium. It is also noteworthy that he appealed specifically to artificial intelligence as an application domain that might benefit from indirect addressing.

REFERENCES

1. Barnden, J.A.
   On short-term information-processing in connectionist theories. *Cognition and Brain Theory, 7* (1), 1984.
2. Barnden, J.A. Diagrammatic short-term information processing by neural mechanisms. *Cognition and Brain Theory, 7* (3&4), 285–328, 1984.
3. Block, N. (Ed). *Imagery.* Cambridge, MA: MIT Press, 1981.
4. Hinton, G.E. & Anderson, J.A. (eds) *Parallel models of associative memory.* Hillsdale, NJ: Lawrence Erlbaum, 1981.
5. Johnson-Laird, P.N. *Mental models.* Harvard University Press: Cambridge, Mass. 1983.
6. Kohonen, T. *Self-organization and associative memory.* Springer-Verlag: Berlin, 1984.
7. Kolers, P.A. Perception and representation. *Ann. Revw. Psychol. 34,* 129–66, 1983.
8. McClelland, J.L., Rumelhart, D.E. and the PDP Research Group. *Parallel distributed processing, Vol. 1: Foundations.* MIT Press: Cambridge, Mass. 1986.
9. Mountcastle, V.B. An organizing principle for cerebral function: the unit module and the distributed system. In G.M. Edelman & V.B. Mountcastle, *The mindful brain.* MIT Press: Cambridge, Mass. 1978.
10. Richardson, J.T.E. *Mental imagery and human memory.* St. Martin's Press: NY. 1980.
11. Rumelhart, D.E., McClelland, J.L. and the PDP Research Group. *Parallel distributed processing, Vol. 2: Psychological and biological models.* MIT Press: Cambridge, Mass. 1986.
12. Sloman, A. Image interpretation: the way ahead? In O.J. Braddick & A.C. Sleigh (eds), *Physical and biological processing of images.* Springer-Verlag: Berlin, 1983.
13. Touretzky, D.S. & Hinton, G.E. Symbols among the neurons: details of a connectionist inference architecture. *Procs. 9th Int. Joint Conf. on Artificial Intelligence,* 1985.
14. Papers in *Cognitive Science, 9* (1), 1985: special issue on connectionism.

N87-26540

NEURAL NETS ON THE MPP

Harold M. Hastings   Stefan Waner
Department of Mathematics
Hofstra University
Hempstead, NY 11550 USA

ABSTRACT

The Massively Parallel Processor (MPP) is an ideal machine for computer experiments with simulated neural nets as well as more general cellular automata. The purpose of this paper is to describe our experiments using the MPP with a formal model neural network. Our results on problem mapping and computational efficiency apply equally well to the neural nets of Hopfield, Hinton et al., and Geman and Geman.

Keywords: Neural Network, Annealing System, Stochastic Cellular Automaton

INTRODUCTION

This paper is a preliminary report on a major component of the research proposal of M. Conrad and the authors, entitled "Applications of stochastic and reaction-diffusion cellular automata." These types of automata are a natural formal setting for theoretical investigations in brain and ecosystem modelling. Most of the proposal was concerned with brain modelling. A significant part of the proposed activity in that area has been completed, and will be discussed here.

Hastings and Pekelney (Ref. 4), observed that many of the properties of the brain seemed to be natural consequences of the working hypothesis that the brain was a large network of McCulloch-Pitts neurons (threshold devices) connected by synapses with stochastic conduction thresholds. In particular, such networks display both gradualism (small changes in inputs cause small changes in outputs (Ref. 1)) and modification-based learning (structural changes as a result of history, Conrad, Ref. 2).

Later, the authors developed a model neural network, implemented the network on a VAX11-780, and conducted experiments in basic learning principles. They also defined (Ref. 5) three postulates which characterized evolutionary learning (for example, by simulated or real neural networks).

Evolutionary Learning

An evolutionary learning system is a formal dynamical system in which the states correspond to modes of information processing, while the suitability of each state is measured by a potential function, the most desirable states possessing least potential. The dynamics of such a system are determined by an annealing process (Refs. 8-9), so that desirable modes are attained by a gradual lowering of the amount of thermal noise.

(The prototype example of an annealing process consists of a gas molecule confined in a potential well, in which the goal is the location of a global potential minimum. If the ambient temperature is lowered sufficiently slowly, the molecule will become trapped in the global minimum with a probability arbitrarily close to one. It is the random behavior of the molecule, "thermal noise," which accounts for its ability to escape from local minima during the cooling process. Simulated annealing then entails simulation of these dynamics in the solution of combinatorically large-scale minimization problems such as the "travelling salesperson"

N87-26540

problem. The essential role played by random noise in such techniques places them outside the realm of algorithmic strategies.)

Further, the potential energy function depends on the environment, so that it is the environment which indirectly determines the equilibria and evolution of the system. We refer to this indirect process of control as soft programming. An evolutionary learning system may then be thought of as a dynamical system which behaves according to three principles:

ergodicity - the use of chaotic behavior to search a state space,

annealing - the regulation of thermal noise by means of (local) lowering of ambient temperature, and

soft programming - the indirect control of the evolution of the system by the environment.

More complex learning regimes were shown to follow the same basic principles (Waner and Hastings, Ref. 10). We also remark that gradualism in annealing systems is a consequence of the annealing dynamics: small changes in the starting point or the shape or potential surface usually cause small changes in the dynamics. Modification of the potential surface through feedback in learning corresponds to Conrad's modification based learning. The annealing dynamics are considered to be internal and inaccessible in detail compared to the feedback dynamics of any learning scheme.

In late 1985 and early 1986 the neural network programs were transported to the MPP. The relationship between the theoretical dynamics and the neural net models will be briefly discussed below.

The rest of this paper is divided into three main parts. The first of these summarizes our neural network models. The second part summarizes our experiments to date on the MPP, and should be understood in the context of a preliminary report. The last part describes conclusions for the application of the MPP and similar massively parallel architectures for our model and similar models. Most of our qualitative conclusions may be readily applied to other neural networks (Hopfield (Ref. 7), Hinton, Sejnowski, and Ackley (Ref. 6), Geman and Geman (Ref. 3)).

THE NEURAL NETWORK

In this section we describe the data structures and algorithms used in our neural network, and briefly describe the dynamics.

Data Structures

The fundamental data structure is a directed graph in which nodes correspond to formal neurons, and arrows to formal synapses. Early experiments on a VAX used a rectangular array of neurons, with nearest neighbor and second-nearest neighbor connections. This structure suggested a natural problem mapping to the MPP. The MPP model uses a 128 x 128 array of formal neurons, with connections to all neighbors in a 5 x 5 array centered at each neuron. This data structure also accords well with a $2+\epsilon$ -dimensional structure for random access in the brain (see Ref. 4 for discussion).

The formal neurons are McCulloch-Pitts neurons. Each contains one or more inputs, has a fixed firing threshold, and fires (sends an output) if and only if the sum of inputs since the last firing is greater than or equal to the firing threshold. The sum of these inputs is called the activity of the neuron; on firing the activity is reset to 0.

The synapses are also threshold devices. Their associated thresholds are called conduction thresholds. However, there are two important differences between the use of thresholds of synapses and those of neurons. First, the conduction threshold of a synapse determines the probability of conduction along that synapse according to the rule prob(conduction) = 1 - (conduction threshold). Second, the conduction thresholds are modified according to two rules:

LEARNING BY REPETITION. Thresholds of synapses which conduct (and similar synapses) are lowered. Conduction thresholds of synapses which do not conduct (and similar synapses) are raised. In the presence of suitable learning regimens, this yields Conrad's modification based learning. The threshold modification scheme must be constructed carefully to minimize the chance of positive feedback in the internal dynamics.

LONG-TERM FORGETTING. Most conduction thresholds (all thresholds except those very near 1 or very near 0) slowly decay to a base value. Thresholds sufficiently close to 0 or 1 do not change; this corresponds to modification-based learning.

Annealing System

Recall that the temperature in an annealing system corresponds to the degree of randomness. In this sense, the entropy (Shannon information) associated with random behavior at all of the synapses corresponds to the temperature. When following a learning regimen results in reducing this entropy, this corresponds to a reduction in temperature. The use of random conduction along synapses yields the underlying diffusion in an annealing system; restricting the dimension to 2 or using a wrap-around topology would guarantee ergodicity. For all practical purposes the present system appears ergodic. Differences in thresholds yield drift terms corresponding to the gradient flow part of annealing. These differences and consequent drift become more pronounced as learning via "annealing through modes of information processing" proceeds.

We remark that classical annealing problems (Ref. 9) can be readily programmed on the MPP with an analogous problem mapping of one cell per processor.

Soft Programming

Soft programming consists of specifying the learning goals. Three types of goals have been studied theoretically. The simplest consists of structured path learning, or learning paths from "sources to "targets." The MPP program below illustrates this case. More complex cases include associative learning and more complex route-finding (only theory so far).

Problem Mapping

We have allocated one processor to each node. This offers the advantages of simple data flow and programming, at the expense of frequently having idle processors in simple experiments.

The Program

```
NEURAL NET:  BEGIN

1.  INITIALIZE
      Net:  initialize thresholds
        initialize activities
        initialize source and
        target, or learning problem

      Supervisory:  initialize random
        number generator, clocks,
        maximum time allowed, etc.

      Learning regimen:  specify.

2.  MAIN LOOP:  REPEAT until timeout
or learning occurs.  Increment random
number generators as necessary
throughout loop.

      AT each neuron:  IF activity is
        greater than or equal to
        firing thresholds, THEN
        FIRE and reset activity to 0.

      AT each synapse, IF neuron at
        tail of synapse has fired, THEN
        TEST for conduction:
          synapse conducts if random
          number is greater than
          conduction threshold.

        IF synapse conducts, THEN
        increment activity of
        neuron at head of synapse.

      AT each synapse, modify
        conduction thresholds incor-
        porating learning by
        repetition and short-term
        forgetting.

      Did net learn?  If so, then
        exit loop and print results.

      Increment clock.

END MAIN LOOP

3.  OUTPUT

END NEURAL NET
```

One should note that the net is intrinsically parallel and stochastic. The parallel feature of the net allows a natural problem mapping: one maps one formal neuron to each processor. Other mappings are possible; for example, one could map each processor to one neuron in a neighborhood of a given neuron, and store the net in an appropriate data structure for transversal. The problem mapping we used was chosen for its simplicity, and potential to reduce the size of the program and maximize computing speed. For example, the MPP program is approximately 20-30% shorter than the VAX program, and both are programmed in similar high-level languages.

The MPP program also ran significantly faster than that of the VAX. The present improvement factor in simple experiments is about 100. However, the MPP does not slow down as the number of neurons firing is increased. This combined with a utilization factor in critical steps of about 5% in simple experiments suggests a relative speed increase in complex tasks should approach 2000.

CONCLUSIONS

Massively parallel architectures are especially appropriate and useful for neural network and similar simulations. In particular, the geometry of the MPP closely parallels the structure of our net model. This places much of the data structure in hardware, reducing computational costs. In addition, much of the computation is "strongly parallel" in the sense that the next state computations must take place simultaneously at many locations. Failing this, data structures and data movement must be developed to simulate this degree of parallelism. Furthermore, most of the VAX computation cost apparently lies in data movement, since no elaborate function evaluations are needed. This contrasts sharply with both algorithms such as Gaussian elimination in which such tight parallelism is not necessary, and algorithms such as many finite element algorithms in which such parallelism is necessary (at least at a simulation level), but in which significant function evaluation costs far exceed data movement costs.

Much of our computing time is spent in random number generation. We are exploring the possibility of realizing random number generators or more general stochastic gates in VLSI hardware. Should this exploration prove successful, it would be possible to construct simple, rapid, and powerful evolutionary learning hardware.

Otherwise, the limited processor power and memory do not slow this type of modelling. In fact, the MPP architecture may well offer the best relative performance because much of the data structure and flow is already present in hardware.

FUTURE DIRECTIONS

At this point, the first part of our proposed research has been largely completed. We have largely developed the theory for applying our learning model to the route-finding problem, and should begin MPP investigations into this problem in early 1987. The extension of these models to more complex (reaction-diffusion) neurons will be done largely by M. Conrad with his former student K. Akingbehin. In a related direction, some successful ecosystem simulations have been performed with more work expected later this year.

REFERENCES

1. M. Conrad, "Evolutionary Learning Circuits," J. Theoret. Biol., 1974, 46, pp. 167-188.
2. M. Conrad, "Complementary Molecular Models of Learning and Memory," BioSystems, 1976, 8, pp. 119-138.
3. S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," Brown University Preprint, 1983.
4. H.M. Hastings and R. Pekelney, "Stochastic Information Processing in Biology," BioSystems, 1982, 15, pp. 155-168
5. H.M. Hastings and S. Waner, "Principles of Evolutionary Learning - Design for a Stochastic Neural Network," BioSystems, 1985, 18, pp. 105-109.
6. G.F. Hinton, T.J. Sejnowski, and D.H. Ackley, "Boltzmann Machines: Constraint Satisfaction Networks that Learn," Cognitive Science, 1984.
7. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Properties," Proc. Nat. Acad. Science USA, 1982, 79, pp. 2554-2558.
8. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," Science, 1983, 220, pp. 671-680.
9. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," J. Chem. Phys., 1953, 21, pp. 1087-1091.
10. S. Waner and H.M. Hastings, "Evolutionary Learning of Complex Modes of Information Processing," in Advances in Cognitive Science, ed. M. Kochen and H.M. Hastings, Amer. Assoc. for the Advancement of Science Selected Symposia, in press.

N87-26541

PARALLEL VLSI ARCHITECTURE EMULATION AND THE ORGANIZATION OF APSA/MPP

John T. O'Donnell
Computer Science Department
Indiana University
Bloomington, IN 47405

ABSTRACT

The Applicative Programming System Architecture combines an applicative language interpreter with a novel parallel computer architecture that is well suited for VLSI implementation. The Massively Parallel Processor can simulate VLSI circuits by allocating one processing element in its square array to an area on a square VLSI chip. As long as there are not too many long data paths, the MPP can simulate a VLSI clock cycle very rapidly. The APSA circuit contains a binary tree with a few long paths and many short ones. A skewed H-tree layout allows every processing element to simulate a leaf cell and up to four tree nodes, with no loss in parallelism. Emulation of a key APSA algorithm on the MPP resulted in performance 16,000 times faster than a Vax. This speed will make it possible for the APSA language interpreter to run fast enough to support research in parallel list processing algorithms.

Keywords: parallel simulation, VLSI, tree architecture, applicative language, functional language.

INTRODUCTION

The Applicative Programming System Architecture research project (Refs. 4, 5, 6) (APSA) combines VLSI hardware design, computer architecture, and programming language research in one unified design for a high level language implementation. The hardware design of APSA uses parallelism extensively, making it impossible to run realistic simulations on sequential processors. This has been a major impediment in the research on APSA. Fortunately, the Massively Parallel Processor (MPP) provides exactly the kind of parallelism needed by APSA's hardware, as well as other parallel VLSI architectures. This paper discusses the goals and structure of the APSA research, and then it describes the methods used for programming the MPP to emulate APSA's parallelism. Finally, it discusses the suitability of the MPP's architecture for this work.

The tasks of designing a programming language implementation, a computer architecture for executing it, and a low level fabrication of the hardware usually proceed independently of each other. As a result, there is a small set of standard basic ways to relate these levels of abstraction to each other: instruction sets provide an interface between language and architecture, while registers, data paths and addressable memories relate the architecture to the hardware. These standards greatly limit the types of architecture and language that are available.

The idea of "high-level language architectures" attempts to improve a computer's performance by breaking away from the standard kind of instruction set, replacing it with a set of instructions especially well suited for a particular programming language. Several manufacturers have built such systems, supporting languages such as Algol, Cobol and Lisp. But these architectures are still limited because they are built with conventional hardware techniques and components.

The Applicative Programming System Architecture research is moving toward improved performance by designing the low level hardware, the high level architecture and the language translator together. Applicative programming languages (also called functional languages) have many advantages over conventional imperative languages, but they are notoriously slow in conventional implementations. The problem is that standard instruction sets give poor support for the basic data structure operations in applicative languages. Furthermore, the standard methods for designing high-level language architectures don't help much, because conventional hardware techniques prevent the architecture from supporting some of the most useful operations.

Research in applicative programming languages is very active, and many examples exist. The current work on APSA is focusing on two particular languages: Scheme (a dialect of Lisp) and SASL. These languages and their relatives play a central role in artificial intelligence, and their use in other fields is expanding. Thus it would be inherently useful to learn how to implement such languages on the MPP.

The APSA architecture contains instructions for manipulating lists, vectors, environments, and continuations — the key data structures for applicative languages. These instructions perform many operations in a constant amount of time which would require iteration in linear time with conventional instructions. For example, consider a program that needs to find the last element of a list. This normally requires a loop, where

PRECEDING PAGE BLANK NOT FILMED

N87-26541

every iteration follows a pointer from one list element to the next. APSA has an instruction that can find the last element of a list in one cycle, and a similar instruction allows indexing into a list to find the *n*th element in 1 cycle. Of course, Fortran can do the same thing with an array on a conventional computer — but Fortran cannot then insert a new element into the middle of the array in constant time. The point is not that APSA supports a fast data structure operation, but rather that it supports many fast operations that can be applied to the same data structure (Ref. 6).

The power of APSA's instruction set results from the ability of its memory to perform parallel logic operations on data, in addition to just storing the data. Each basic storage unit in the memory, called a cell, can hold one word of data. An APSA word corresponds roughly to a Lisp cons box or a Fortran array element. The cells are organized into a linear address space, just as in conventional computer memories. However, each cell also contains low-level processing capability. In addition, there are two basic kinds of internal memory operation that require parallel hardware; these operations lead to the differences between APSA memory and conventional memory.

- *Shift.* Each cell reads the contents of its left or right neighbor, performs a logic operation on that value and its current state, and stores a new value — which may be its old value, the contents of its neighbor, or another value (resulting from a sweep operation). If a number of adjacent cells all store the contents of their left neighbor, the net effect is that a sequence of words shifts right. Since each cell's logic hardware controls the value that it stores, some cells may do a shift while others remain unchanged. This allows the memory to insert or delete a word in the middle of a data structure, in one clock cycle.
- *Sweep.* The memory performs a global logic operation on the contents of all the cells and an input from the memory controller. Part of the logic hardware computes a value that it returns to the controller; this is how the controller is able to fetch data from the memory. Other parts of the logic compute independent values to be sent to each cell in the memory. If we ask for arbitrary logic operations, the complexity of the hardware would get out of hand. Therefore, APSA supports only logic operations that can be implemented with a binary tree of combinational logic components. This restriction leads to manageable but powerful hardware.

The shift operation requires data paths connecting each pair of adjacent cells, and the sweep operation requires a binary tree of combinational logic whose root is a port to the memory controller and whose leaves are the cells. Figure 1 illustrates this organization.

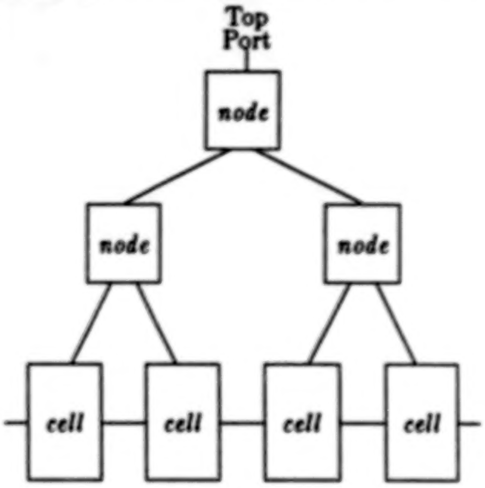


Figure 1. APSA Memory Organization

The reason that APSA algorithms cannot be used in implementing applicative languages on ordinary computers is simply that simulating the shift and sweep operations takes too much time. At the very least, a computer needs several thousand words of memory to be able to run interesting programs. But that would require a sequential simulator to operate separately on several thousand cells and several thousand logic tree nodes — just to compute the effect of APSA's memory during one clock cycle. APSA's algorithms are fast, but they are not fast enough to overcome this speed penalty of 3 to 6 orders of magnitude.

VLSI technology (very large scale integrated circuits) is good at implementing highly parallel systems with large numbers of small components, as long as the data paths connecting the components do not waste too much area on the chip. There is a standard method for laying out a binary tree on a square VLSI layout, where the subtrees appear on opposite sides of the node above them. For nodes at an even-numbered level, the subtrees are placed to the east and west; nodes at an odd-numbered level have their subtrees placed to the north and south. This results in the "H-tree layout" (Ref. 3). It also turns out that adjacent cells in an H-tree can be connected efficiently.

The original idea in the APSA project was to exploit the properties of VLSI in order to build a parallel data structure memory. This VLSI design is currently in progress, but there is another problem: details of the design of the cells and the logic tree have a profound impact on the large scale performance of the applicative programming language. For example, a low level decision on the representation of data in a cell may affect the frequency of garbage collections. Furthermore, the system must run for thousands of cycles before such effects show up. This has lead to a very serious difficulty: simulation of the VLSI layout is far too slow to allow enough experimentation with the system's overall behavior to be able to make correct decisions about details of the VLSI layout!

The MPP provides exactly the same kind of parallelism that VLSI does. Each processing element (PE) combines storage with logic, and is connected to nearby PEs. Thus it is possible to map a square VLSI layout onto the square MPP array, with each PE performing the function of the corresponding area of the chip. The MPP also has a limitation: long distance communications are relatively slow because messages must be sent along a path consisting of adjacent PEs. Long data paths also lead to poor performance in VLSI, so good chip designs tend to have a small number of long data paths and a very large number of short ones. Such designs are well suited for implementation on the MPP.

Ref. 7 describes the basic methods of digital circuit simulation and outlines how the MPP can simulate VLSI circuits. The next section discusses APSA data structure operations and parallelism, and the remainder of this paper is concerned specifically with the implementation of APSA on the MPP. The APSA/MPP program directly transfers all of APSA's parallelism into MPP parallelism, so it seems more appropriate to call it an emulator rather than a simulator.

DATA STRUCTURES IN APSA

A complete description of the data structures and algorithms for APSA is beyond the scope of this paper, but a brief example will clarify the general ideas. The most interesting algorithms are for maintaining aggregate data structures, environments, continuations and for performing garbage collections. These algorithms are too complex to discuss here, so this section describes operations on a simple combined list/vector data structure.

APSA can represent the list `(a b d e)` in a compact form by storing the list elements in consecutive neighboring cells. Each cell contains a type field; we are only concerned with the cells that contain a *value*. In addition, each cell contains several flags. The *attached* flag (denoted by ▷) indicates that a cell represents a value in a list that continues on into the next cell. Therefore all the list elements except the last will be in cells with ▷ set. This leads to the following representation:

	▷	▷	▷		
	val	*val*	*val*	*val*	
	a	b	d	e	

Now suppose that we want to insert a new element "c" into the list just after the occurrence of "b". This takes three instructions. First, the program must locate the point where the insertion should be made, using an associative search. The *match* instruction does this, and sets the *select* flag (denoted by •) in that cell:

	▷	• ▷	▷		
	val	*val*	*val*	*val*	
	a	b	d	e	

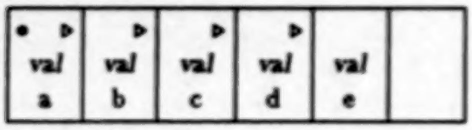
As the program inserts c into the list, it must move all the elements before it to the left, leaving room for the insertion. This will destroy the contents of the leftmost cell, which is part of the available space list. However, only the cells to the left of the point of insertion should store the contents of their right neighbor; cells to the right of the insertion must remain unchanged. Therefore the program must compute another flag called *mark* (denoted by ∘) which will control the shift. The *mark to select* instruction sets ∘ in all the cells that lie to the left of the selected cell:

∘	∘ ▷	• ▷	▷		
	val	*val*	*val*	*val*	
	a	b	d	e	

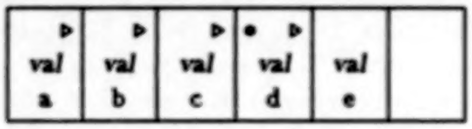
Now the program can issue the *insert (c)* instruction, which causes each cell to perform an operation that depends on its flag settings. Cells with ∘ set will store the contents of their right neighbor, the cell with • set will store the instruction operand c, and all other cells remain unchanged. The net effect is to insert c into the list, while destroying one element of the available space pool:

∘ ▷	∘ ▷	• ▷	▷		
val	*val*	*val*	*val*	*val*	
a	b	c	d	e	

Note that such an insertion always takes three cycles, regardless of where the insertion takes place. List processing systems based on linked representation can also quickly insert a new element. However, suppose that the program now needs to find the fourth element in the list. The linked list representation requires iteration to do this; the time to find the *n*th element is proportional to *n*. APSA can index directly to the *n*th list element, just as if the list is an ordinary vector. First, the program uses the *match* instruction again to select the beginning of the list:



Next, the *index (n)* instruction directly locates the *n*th element in one cycle. Thus *index (3)* produces:



The point of this example is that *one data structure supports both vector and list operations.* This leads to a richer set of data structures and algorithms than conventional computers provide. Notice that APSA used parallelism inside each instruction, allowing it to perform a task using fewer instructions than a conventional system.

APSA uses more complex hardware to execute fewer instructions. This leads to a crucial question: *does APSA's speedup in number of cycles overcome its overhead in cycle time?* Consider first the speedup. Indexing to the *n*th element of a list normally takes time $O(n)$, while APSA does this in time $O(1)$. Many other APSA algorithms show the same $O(n)$ reduction in the number of instructions executed. Calculating the overhead in cycle time is a more subtle problem. At first sight, it appears that APSA's cycle time is slower than a conventional machine's by a factor of $O(\log n)$ because of the tree. We normally think of a computer's RAM memory as having a constant access time: as the problem size grows the memory does not slow down. However, that is only true as long as the memory is large enough to hold the problem. And it turns out that as the size of a RAM memory increases, its access time slows down. There are two distinct reasons for this. First, a RAM uses a decoder to select the addressed word — *and a decoder is a tree of combinational logic.* Second, as a RAM grows in size its wires become longer, and the electrical delay across the wires becomes significant. Of course, both of these factors affect APSA: it contains log time delay in its tree, and as its memory grows its data paths become longer, requiring more communication time. The final result is that

- the APSA memory has the same asymptotic cycle time as a RAM memory, although it is slower by a constant factor K, and
- fewer cycles are needed to run an algorithm on APSA than on a conventional machine.

Table 1 compares the cycle time for RAM vs. APSA using three different measures: counting one time unit per cycle, considering the delay through the combinational logic trees, and considering also the electrical wire delay.

measure	*RAM*	*APSA*
constant	1	1
logic delay	$\log n$	$\log n$
wire delay	$\sqrt{n}$	$\sqrt{n}$

Table 1. Cycle time complexity

An APSA cycle is slower than a RAM's by a constant factor K, which determines the actual attainable speed. Fortunately, the MPP sheds light on the value of K. The MPP contains a tree of logic that computes the logical *or* of the P register in all 16,384 PEs. This corresponds roughly to a 1-bit upsweep, and it takes about half a microsecond. The APSA instructions require several bits going both up and down, but it should still be feasible to attain a hardware cycle time on the order of 10 microseconds, which would lead to excellent performance. However, the emulator for APSA that runs on the MPP takes several hundred microseconds per cycle. This is good enough for extensive experimentation, but will probably not be competitive with Lisp implementations on conventional computers.

The preceding discussion concerned parallelism within data structure operations. There is also another level of parallelism. The APSA memory can perform a parallel data structure operation on several different data structures — in parallel. For example, the Lisp **mapcar** functional applies a function to every element in a list. In some cases, APSA can execute all these applications simultaneously. For example, the *index* operation can index into many lists at the same time. However, storage allocation can lead to problems. If a program tries to perform many **cons** operations at the same time, the system may require a number of cycles to obtain all the new storage words that it needs. Further work is needed to assess the potentials and limitations of this form of parallelism.

A promising method for implementing applicative programming languages is to translate them into combinators, which can then be reduced on an appropriate architecture. APSA has the ability to reduce many combinators simultaneously, although a number of problems with storage allocation remain. This looks like a good approach for designing parallel combinator reduction machines.

THE SKEWED TREE LAYOUT

The basic idea in mapping the VLSI memory design onto the MPP array unit is to simulate an area of the chip with each processing element. If that is done in a straightforward manner, only 1/4 of the PEs will

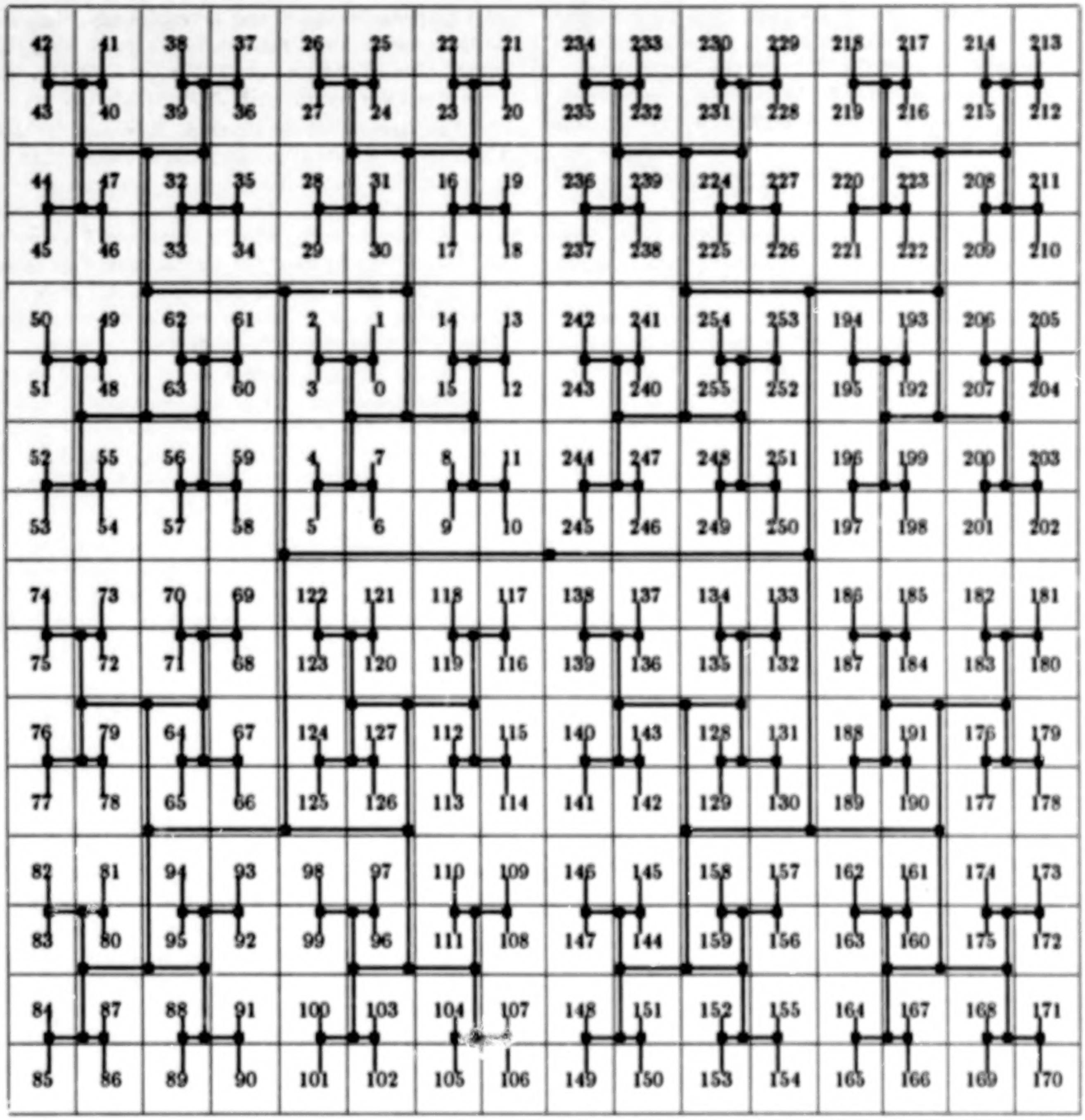

42 41 38 37 26 25 22 21 234 233 230 229 218 217 214 213
43 40 39 36 27 24 23 20 235 232 231 228 219 216 215 212
44 47 32 35 28 31 16 19 236 239 224 227 220 223 208 211
45 46 33 34 29 30 17 18 237 238 225 226 221 222 209 210
50 49 62 61 2 1 14 13 242 241 254 253 194 193 206 205
51 48 63 60 3 0 15 12 243 240 255 252 195 192 207 204
52 55 56 59 4 7 8 11 244 247 248 251 196 199 200 203
53 54 57 58 5 6 9 10 245 246 249 250 197 198 201 202
74 73 70 69 122 121 118 117 138 137 134 133 186 185 182 181
75 72 71 68 123 120 119 116 139 136 135 132 187 184 183 180
76 79 64 67 124 127 112 115 140 143 128 131 188 191 176 179
77 78 65 66 125 126 113 114 141 142 129 130 189 190 177 178
82 81 94 93 98 97 110 109 146 145 158 157 162 161 174 173
83 80 95 92 99 96 111 108 147 144 159 156 163 160 175 172
84 87 88 91 100 103 104 107 148 151 152 155 164 167 168 171
85 86 89 90 101 102 105 106 149 150 153 154 165 166 169 170

Figure 2. Skewed H-tree layout

simulate a memory cell; the others will be dedicated to the tree nodes and the data paths connecting the nodes, while some of the PEs would be wasted entirely. The original VLSI layout for APSA had these properties (Ref. 8). There is nothing inherently wrong with this approach, but it is possible to go from a 4,096 cell layout to a 16,384 cell layout without losing any potential parallelism.

By placing one memory cell in each PE, the emulator can perform all the cell logic operations in parallel. The nodes never execute in parallel with the cells anyway, because of data dependencies in the sweep algorithm. This means that we lose no parallelism if each PE simulates a node in addition to a cell. Furthermore, the nodes execute simultaneously only at one level in the tree at a time. For example, on an upsweep all the cells execute, then all the bottom level nodes, then all the nodes at the next higher level, etc. Therefore a PE can simulate a cell and several nodes, as long as all of its nodes appear at different levels in the tree.

These considerations lead to a "skewed H-tree" layout that allows all 16,384 PEs to contain a cell. Each data path between two subtrees is skewed to one side or the other, so that its node and path elements actually lie on top of one of the subtrees (forcing some PEs to hold several nodes). We place the node and path on whichever side yields the shorter data paths (and hence the greatest speed).

Figure 2 shows the skewed layout for a 16×16 array, with 256 cells. Each cell is represented by its address (the leftmost cell is 0 and the rightmost is 255). Small black squares indicate nodes, while the data paths connecting nodes appear as thick lines. Thin lines show the PE array, making it easy to see what cells, nodes and paths lie in any PE.

This layout consists of a recursive sequence of squares. A *square* is a structure with

- a *central node* near the center (i.e., in one of the four PEs adjoining the true geometric center),
- data paths from the central node going east and west to the *eastern node* and the *western node* respectively, and
- data paths leading from the eastern and western nodes to the *northern subtrees* and the *southern subtrees*.

For example, consider the 2×2 square at the northwest corner of Figure 2, consisting of cells 40–43. The central node is at level 2 in the tree (there are 2^2 cells below it) and it happens to be in the PE to the southeast of the geometric center of the 2×2 grid: thus this node has *SE parity*. The eastern and western nodes, both of level 1, appear in cells 40 and 43 respectively. Since the northern and southern subtrees are at level 0, they just consist of cells. Notice that 2×2 squares appear with all four possible parities: NE, NW, SE and SW.

The largest square (level 8, in cell 245) illustrates several points that affect the communication algorithms. The eastern node has W parity, in order to minimize its data path. For the same reason the western node has E parity. Both of these level 7 nodes inherit their N parity from the central node. One of the data paths from a node to its subtrees will always be shorter than the other one — and the communication algorithms must be able to handle this.

The skewed H-tree has a useful property: no PE represents more than four nodes, regardless of the size of the layout. That is significant because every PE must allocate memory for one cell and all its nodes. Each PE contains only 1024 bits, so an unbounded number of nodes per PE would take too much space.

The sweep algorithm needs to know how long the tree data paths are in order to control its shifting loops. A simple recurrence equation yields the length d_i of a long path from a node at level i down to its subtrees (the short path has length $d_i - 1$):

$$d_1 = 1 \qquad d_2 = 1 \qquad d_3 = 1 \qquad d_4 = 1$$

$$d_i = d_{i-1} \quad \text{for } i \text{ even, } i > 4$$

$$d_i = 2d_{i-1} \quad \text{for } i \text{ odd, } i > 4.$$

The sweep algorithm computes these recurrences as it moves up or down the tree. Thus it must periodically multiply the current d by 2 on an upsweep and divide d by 2 on a downsweep. The sweeps run entirely in the PE Control Unit, without any help from the Main Control Unit, and the PECU has no scalar arithmetic facilities — so the sweep algorithm does its multiplications and divisions using Peano arithmetic, completely in parallel with other operations!

It is possible to place data paths connecting neighboring cells in the existing tree structure. Using that method, a shift operation would send data from a cell part way up the tree, and then back down the other branch of the tree to the neighbor. This is probably the best way to implement the cell neighbor data paths in a VLSI layout, because it reduces the number of wires that must cross each other. However, it is faster on the MPP to route data in a straight line from each cell to its neighbor, and the APSA emulator takes this approach.

The shift algorithm operates by sending the contents of each cell to its neighbor within one layout square. It repeats the process for each square, working from the smallest square size (2×2) up to the largest (normally 128×128). At each square size, all squares

d k	*sequence of operations*				
0 1			*rw*		
0 2			*rw rw*		
0 3			*rw rw rw*		
0 4			*rw rw rw rw*		
1 1	*r*	→	*s*	→	*w*
1 2	*r sr*		→		*w sw*
1 3	*r sr*	→	*wsr*	→	*w sw*
1 4	*r sr*	→	*wsr wsr*	→	*w sw*
1 5	*r sr*	→	*wsr wsr wsr*	→	*w sw*
1 6	*r sr*	→	*wsr wsr wsr wsr*	→	*w sw*
2 1	*r*	→	*s s*	→	*w*
2 2	*r sr*	→	*s*	→	*w sw*
2 3	*r sr sr*		→		*w sw sw*
2 4	*r sr sr*	→	*wsr*	→	*w sw sw*
2 5	*r sr sr*	→	*wsr wsr*	→	*w sw sw*
2 6	*r sr sr*	→	*wsr wsr wsr*	→	*w sw sw*
2 7	*r sr sr*	→	*wsr wsr wsr wsr*	→	*w sw sw*
3 1	*r*	→	*s s s*	→	*w*
3 2	*r sr*	→	*s s*	→	*w sw*
3 3	*r sr sr*	→	*s*	→	*w sw sw*
3 4	*r sr sr sr*		→		*w sw sw sw*
3 5	*r sr sr sr*	→	*wsr*	→	*w sw sw sw*
3 6	*r sr sr sr*	→	*wsr wsr*	→	*w sw sw sw*
3 7	*r sr sr sr*	→	*wsr wsr wsr*	→	*w sw sw sw*
3 8	*r sr sr sr*	→	*wsr wsr wsr wsr*	→	*w sw sw sw*
4 1	*r*	→	*s s s s*	→	*w*
4 2	*r sr*	→	*s s s*	→	*w sw*
4 3	*r sr sr*	→	*s s*	→	*w sw sw*
4 4	*r sr sr sr*	→	*s*	→	*w sw sw sw*
4 5	*r sr sr sr sr*		→		*w sw sw sw sw*
4 6	*r sr sr sr sr*	→	*wsr*	→	*w sw sw sw sw*
4 7	*r sr sr sr sr*	→	*wsr wsr*	→	*w sw sw sw sw*
4 8	*r sr sr sr sr*	→	*wsr wsr wsr*	→	*w sw sw sw sw*
4 9	*r sr sr sr sr*	→	*wsr wsr wsr wsr*	→	*w sw sw sw sw*

Table 2. *The pattern of communications for send*

in the system participate simultaneously in the shift, which sends data between the four *corners* of the square. For example, at the 2×2 square size in Figure 2, a shift right operation (moving the contents of cell i to the neighbor input field of cell $i+1$) causes the contents of cell 40 to move north to 41, while cell 41 moves west to 42, 42 moves south to 43, and 43 moves east to 40. All other 2×2 squares perform this pattern of moves at the same time.

In general, the four data movements (north, west, south and east) will include three legitimate movements and an invalid one. Thus it was invalid to move the contents of cell 43 east to 40. However, the particular direction of movement that is invalid depends on the parity of the square's central node. Since central nodes appear with each of the four parities NE, NW, SE, SW, each of the four data movements will be invalid for 1/4 of the squares.

Fortunately, the invalid fields left by data movement at one square level are all overwritten when the shift algorithm does the next higher level. After doing the 2×2 squares, the corner cells of all the 4×4 cells send their data to their neigbors (within the square). For the 4×4 square in the northwest corner of Figure 2, the four corner source cells are 35, 39, 43 and 47 — so the west move writes the contents of cell 39 into cell 40, replacing its old invalid contents left by the 2×2 moves. Of course, 1/4 of the moves at this level are also invalid, but when the shift algorithm reaches the highest square level (128×128), the only cell that remains with an invalid neighbor input field is the leftmost cell in the entire memory: cell 0 holds the contents of cell 255. If the shift algorithm was called to rotate the memory, this is the desired effect. Otherwise, after handling the highest level square, the shift algorithm writes a value into cell 0 that it receives from the memory controller.

THE COMMUNICATION ALGORITHMS

The sweep and shift algorithms must move k bits (called the *data path width*) from a set of *source PEs* to the corresponding *destination PEs*. The preceding section outlined how shift and sweep locate the sources and destinations and calculate the path distances d for each move. It is also important to consider the details of the data movement algorithm, *send*, because this is the innermost loop of the APSA emulator, accounting for much of the total execution time.

One factor simplifies the implementation of send: all of the movements that it must perform at once go in a straight line for the same distance d. However, two other factors complicate it:

- Simultaneous movements may go in different directions. In particular, sweep always needs to send data east and west, or north and south. The shift algorithm always sends data all four directions.
- The two key parameters of a send — the distance d and the number of bits k — are variables computed by the calling algorithm. Therefore the send code must be flexible enough to handle automatically any combination of d and k.

Sending a bit from the source to the destination involves three basic operations:

r *read* the bit into the source PE;

- *s* *shift* each bit on the data path one PE closer to the destination; and
- *w* *write* the bit into the destination PE.

Since all the data paths have the same length and width, and none of them cross each other, we can concentrate on what happens to the data during a single movement.

Any send operation consists of a pattern of the *r*, *s* and *w* primitives. For example, *rsssw* will send a single bit (k=1) along a path 3 PEs long (d=3). The situation is more complex when $k > 1$. In order to minimize the communication time, it is essential to pipeline data along the path. This leads to a number of distinct patterns of communication, illustrated by Table 2. The send algorithm must use the values of d and k to determine the best communication pattern.

All the sources, destinations and data paths necessary for sweep and shift must be marked in advance by an initialization algorithm. The initializer's main function is to compute a set of *mask lists*, one for every node level in the tree. Every processing element has its own set of mask lists, and each one contains 14 individual masks that indicate whether the PE contains a node at each level in the tree, what the nodes' parities are, and what data paths pass through the PE. The mask lists allow the *r*, *s* and *w* primitives to operate in the relevant PEs without disturbing data in the others. For example, the *r* primitive places a "node present" mask in the G register, and then executes a `loada P,SOURCE` instruction.

ORGANIZATION OF THE EMULATOR

The APSA system breaks naturally into three components, and the MPP implementation reflects this structure by running the components on separate computers:

- The memory instruction set, which provides the heart of APSA's parallelism and which requires simulation of a VLSI architecture layout, runs entirely in the Processing Element Control Unit (PECU). This software, written in Pearl, includes the initializer, shift and sweep algorithms, as well as the node and cell logic functions for each instruction. The speed of these programs is of paramount importance for the overall goals of the research.
- The memory controller and applicative language interpreter, written in MCL, reside in the Main Control Unit (MCU), and issue instructions to the APSA memory through the call queue.
- The I/O system, written in Fortran, runs on the MPP host.

PERFORMANCE RESULTS

This section discusses the performance of the emulator's initializer and sweep algorithm. The shift algorithm is similar to sweep, but it will run considerably slower because its data path width will usually be around 64 bits, while typical sweeps only require 3 or 4 bits.

The MPP execution times were measured using the performance monitor, which is extremely accurate and repeatable. In addition, the MPP simulator predicted the same performance times for the smaller APSA layouts.

All of the emulation algorithms (initialize, sweep and shift) take a parameter that specifies the size of the APSA layout — they don't depend on the 128×128 size of the MPP array unit. In general, the layout may be $n \times n$ for any n that is a power of 2. Consequently, the height of the binary tree may be any even number.

Table 3 shows the time required to initialize APSA for all the layouts that fit in the MPP array. While growing a layout the initializer must perform one complete shift and five complete downsweeps, in addition to performing several other functions. The *Layout* column shows the dimensions of each APSA layout. The *Level* column gives the height of the binary tree for each layout; thus a layout with *Level*=n will contain 2^n *Cells*. The *Square* column shows the depth of recursion in the H-tree layout, which is also the number of iterations in the outer loops of the sweep and shift algorithms.

Square	*Level*	*Layout*	*Cells*	*Time (μs)*
1	2	2×2	4	270
2	4	4×4	16	429
3	6	8×8	64	596
4	8	16×16	256	784
5	10	32×32	1,024	1,019
6	12	64×64	4,096	1,339
7	14	128×128	16,384	1,817

Table 3. APSA initialization times in μs (MPP)

Table 4 gives the MPP execution times of the upsweep algorithm for data paths that are 1, 2 and 3 bits wide. (The downsweep algorithm is almost identical.) There are three phenomena that account for the variations among these times. First, the number of iterations of the main sweep loop and the number of calls to the node logic function both depend on the layout size (the *Square* column in Table 3 gives these values). Second, the data paths become longer toward the top of the tree in larger layouts — that is way the execution time grows faster than linearly as a function of the *Square* size. Third, the width of the data path (i.e.,

Layout	*1 Bit*	*2 Bits*	*3 Bits*	*Increase*
2×2	12	17	23	
4×4	20	31	42	+19
8×8	29	45	62	+20
16×16	39	62	85	+23
32×32	51	80	110	+25
64×64	68	104	140	+30
128×128	95	137	179	+39

Table 4. Upsweep times in μs (MPP)

Layout	*MPP*	*Vax 780*	*Speedup*
2×2	23	610	27
4×4	42	2,[illegible]0	60
8×8	62	10,500	169
16×16	85	53,000	624
32×32	110	175,000	1,591
64×64	140	706,000	5,043
128×128	179	2,872,000	16,045

Table 5. Comparison of upsweep times in μs

the number of bits being sent up the tree) determines the number of iterations of sweep's inner loop, and the node logic function also requires more time to operate on more bits.

At this point it is interesting to consider the implications of Table 4 on the future course of the APSA research project. Serious development of parallel Lisp and SASL will require a moderately large memory. That is why the emulator supports a skewed H-tree instead of the conventional H-tree: it can place a memory cell in every MPP processing element, providing the largest memory size that is possible without loss in parallelism. Therefore we are primarily interested in the last line of Table 4, which gives sweep times for a 16,384 cell memory. Assuming two sweeps and several parallel operations in all the cells, a typical APSA instruction should take about half a millisecond. This is certainly adequate for extensive experimentation and development. There is also the possibility that some applicative algorithms may by able to perform the equivalent of several hundred operations simultaneously. If that happens, the MPP/APSA system could partially achieve a long-standing goal of the programming language research community: automatically speeding up a list processing program on a parallel machine, *without requiring the programmer to specify any parallel operations* (Refs. 11, 12). Of course, this goal is still far in the future.

Table 5 illustrates what the future of APSA would be like without the MPP: it repeats the upsweep times for a 3 bit data path and compares them with an APSA upsweep algorithm running on a Vax 780 computer. The Vax program is written in C, and it exploits some of the standard techniques for efficiency in C (for example, it increments pointers into arrays in order to avoid most array index calculations). These timings were generated using the "time" command in Berkeley Unix 4.2BSD, and there is an error range of about ±10%. The columns headed *MPP* and *Vax 780* give the running time in microseconds of a 3 bit upsweep on each layout size. The *Speedup* column gives the ratio of *Vax 780* time divided by *MPP* time. It is important to realize that these figures do *not* show how much faster the MPP is than a Vax. They merely indicate how much the performance of upsweep can be improved by moving from a Vax to the MPP.

For the case of most interest — a 3-bit upsweep on a full 128×128 layout — the MPP requires 179 μs, while the Vax takes 2.87 seconds. At this rate, one MPP-minute of upsweeps would correspond to eleven Vax-days of CPU time. Thus the MPP makes it easy to run emulations that would be inconceivable using conventional computers.

DISCUSSION

The speedups in Table 5 may appear surprising, because the MPP is nominally only on the order of 512 times faster than a Vax (it has 16,384 times more processors and its word size is 1 bit, compared with 32 on the Vax). But the actual performance speedup depends in detail on the interaction between the algorithm and each computer's architecture. In particular, the upsweep algorithm is inherently bit serial, so the large Vax word length does not help at all. The shift algorithm uses long words, so the MPP's speedup will be much smaller than for sweep.

It would be extremely valuable to investigate the performance of APSA on a Connection Machine (Ref. 2) and compare it with the results given above. Since the Connection Machine contains a network with long data paths, it will be faster than the MPP for sufficiently large layouts. On the other hand, the MPP's faster cycle time should make it faster for smaller layouts. Table 4 shows that much of the time for an upsweep for a 128×128 layout goes into bit-serial operations and the node logic functions, where the MPP is faster. The longest data paths have length 32; the MPP is probably still faster than the Connection Machine at this size because of its synchronous communications and fast clock.

There are several other approaches to parallel implementation of Lisp and related languages. Multilisp (Ref. 1) gives the programmer a parallel construct called a *future*, which allows parallel evaluation of independent expressions. Multilisp is being used to program an MIMD parallel computer. Connection Machine LISP (Ref. 10) takes an approach similar to APSA: it makes parallel data structures available to the user programmer. However, it does not integrate parallel data structures as deeply into the interpreter's environment and continuations as APSA does. Another method for implementing car and cdr on the MPP (Ref. Potter) allows parallel searching of many lists, but its cons function requires time and space proportional to the length of the second argument.

CONCLUSION

The Applicative Programming System Architecture research is concerned with designing an applicative language, a computer architecture and a VLSI hardware implementation together, so that they cooperate effectively. This research is leading toward better language implementations and new architecture designs. However, simulating the low level hardware on a conventional computer is too slow to allow experimentation with APSA's parallel algorithms.

The Massively Parallel Processor is ideal for simulating VLSI circuits that have regular designs and short data paths, such as systolic arrays. The VLSI layout for APSA's memory contains parallel logic in every memory cell and a binary tree for communications and additional logic. The tree layout contains only a few long paths, so it is well suited for implementation on the MPP. The key tree communication algorithm shows a huge speedup — by a factor of 16,000 — compared with simulation on a conventional computer.

The basic APSA operations are fast enough on the MPP to allow experimentation with a realistic parallel implementation of an applicative programming language, and a parallel implementation of Lisp is in progress. Thus the MPP is making it possible to study new ideas in parallel VLSI architectures.

ACKNOWLEDGEMENT

I would like to thank Judy Devaney for helping me with the I/O macros in MCL, and especially for maintaining the simulator, which is extremely useful for developing large Pearl programs.

REFERENCES

1. Robert H. Halstead, Jr., "Multilisp: A Language for Concurrent Symbolic Computation", *Trans. on Prog. Lang. and Systems*, pp. 501-538, Vol. 7, No. 4, Oct. 1985.

2. W. Daniel Hillis, *The Connection Machine*, Cambridge: The MIT Press, 1985.

3. Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.

4. John T. O'Donnell, *A Systolic Associative LISP Computer Architecture with Incremental Parallel Storage Management*, Technical Report 81-5, Computer Science Department, University of Iowa, Iowa City, 1981.

5. John T. O'Donnell, "An Architecture that Efficiently Updates Associative Aggregates in Applicative Programming Languages", *1985 IFIP Symposium on Functional Programming Languages and Computer Architecture*, Nancy, France, September 1985, *Lecture Notes in Computer Science 201*, pp. 164-189, New York: Springer-Verlag.

6. John T. O'Donnell, "An efficient architecture for implementing sparse array variables," *Proceedings of the Twenty-third Allerton Conference on Communication, Control and Computing*, pp. 986-995, Coordinated Science Laboratory, University of Illinois, October, 1985.

7. John T. O'Donnell, "Finely Grained Parallelism in an Applicative Architecture", *Proceedings of the Workshop on Future Directions in Computer Architecture and Software*, pp. 372-374, Army Research Office, May, 1986.

8. John T. O'Donnell, "Simulating VLSI Systems Using the Massively Parallel Processor", *Proceedings of the 1986 Summer Computer Simulation Conference*, pp. 153-158, The Society for Computer Simulation, July, 1986.

9. Jerry L. Potter, "List Based Processing on the MPP", in *The Massively Parallel Processor*, (ed. J. Potter), Cambridge: The MIT Press, 1985.

10. Guy L. Steele, Jr., W. Daniel Hillis, "Connection Machine LISP: Fine-Grained Parallel Symbolic Processing", pp. 279-297, *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*.

11. Philip C. Treleaven, "Computer Architecture for Functional Programming", pp. 281-306 in *Functional Programming and its Applications* (ed. J Darlington), Cambridge University Press, 1982.

12. Steven R. Vegdahl, "A Survey of Proposed Architectures for the Execution of Functional Languages", *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984.

INVITED PAPER

Dr. Stephen Wolfram
Center for Complex Systems Research
University of Illinois

N87-26542

Cellular Automaton Supercomputing*

Stephen Wolfram
Center for Complex Systems Research,
and Departments of Physics, Mathematics and Computer Science,
University of Illinois at Urbana-Champaign, 508 South Sixth Street,
Champaign, IL 61820.

Many of the models now used in science and engineering are over a century old. And most of them can be implemented on modern digital computers only with considerable difficulty. This paper discusses some new basic models which are much more directly suitable for digital computer simulation.

The ultimate purpose of most scientific investigations is to determine how physical or other systems will behave in particular circumstances. Over the last few years, computer simulation has been emerging as the most effective method in many different cases. The basic approach is to use an algorithm which operates on data in the computer so as to emulate the behaviour of the system studied (e.g. [1]). This algorithm can be considered to provide a "computational model" for the system.

Theoretical investigations of physical systems have conventionally been based on a few definite classes of mathematical models. By far the most common are partial differential equations (e.g. [2]). These equations were designed to describe systems such as fluids which can be considered as continuous media. Calculus was used as a tool to find mathematical formulae for the solutions to these equations. This allowed great progress to be made in the understanding of many phenomena, particularly those such as electromagnetism, which are by linear partial differential equations. Progress was also made in studies of processes such as laminar (regular) fluid flows, which can be approximated by linear partial differential equations. But the standard methods of mathematical analysis made little headway on problems such as fluid turbulence, for which non-linear partial differential equations are essential.

When digital computers became available, it was natural that they should be used to try and find solutions to such partial differential equations. But digital computers can represent such equations only approximately. While equations involve continuous variables, digital computers can treat only discrete, digital, quantities. The real numbers which correspond to continuous variables in the equations must be represented on the computer by packets of bits, typically in the form of 32 or 64 bit numbers in floating-point format. In addition, the derivates which appear in the equations must be approximated by finite differences on a discrete grid. Much effort has been spent in numerical analysis to show for example that with sufficiently fine grids, exact solutions to the continuum equations can be found. Unfortunately, such theorems have been proved almost exclusively only in cases where exact solutions to the continuum equations are known. For most important non-linear equations, quite ad hoc methods must be used to gauge the accuracy of approximations.

Nevertheless, the thrust in scientific computation has been to develop computer hardware and algorithms which allow more and more extensive approximations to partial differential

---

* Also to appear in *High Speed Computing: Scientific Applications and Algorithm Design*, edited by Robert B. Wilhelmson, (University of Illinois Press, 1987).

PRECEDING PAGE BLANK NOT FILMED

ORIGINAL PAGE IS
OF POOR QUALITY

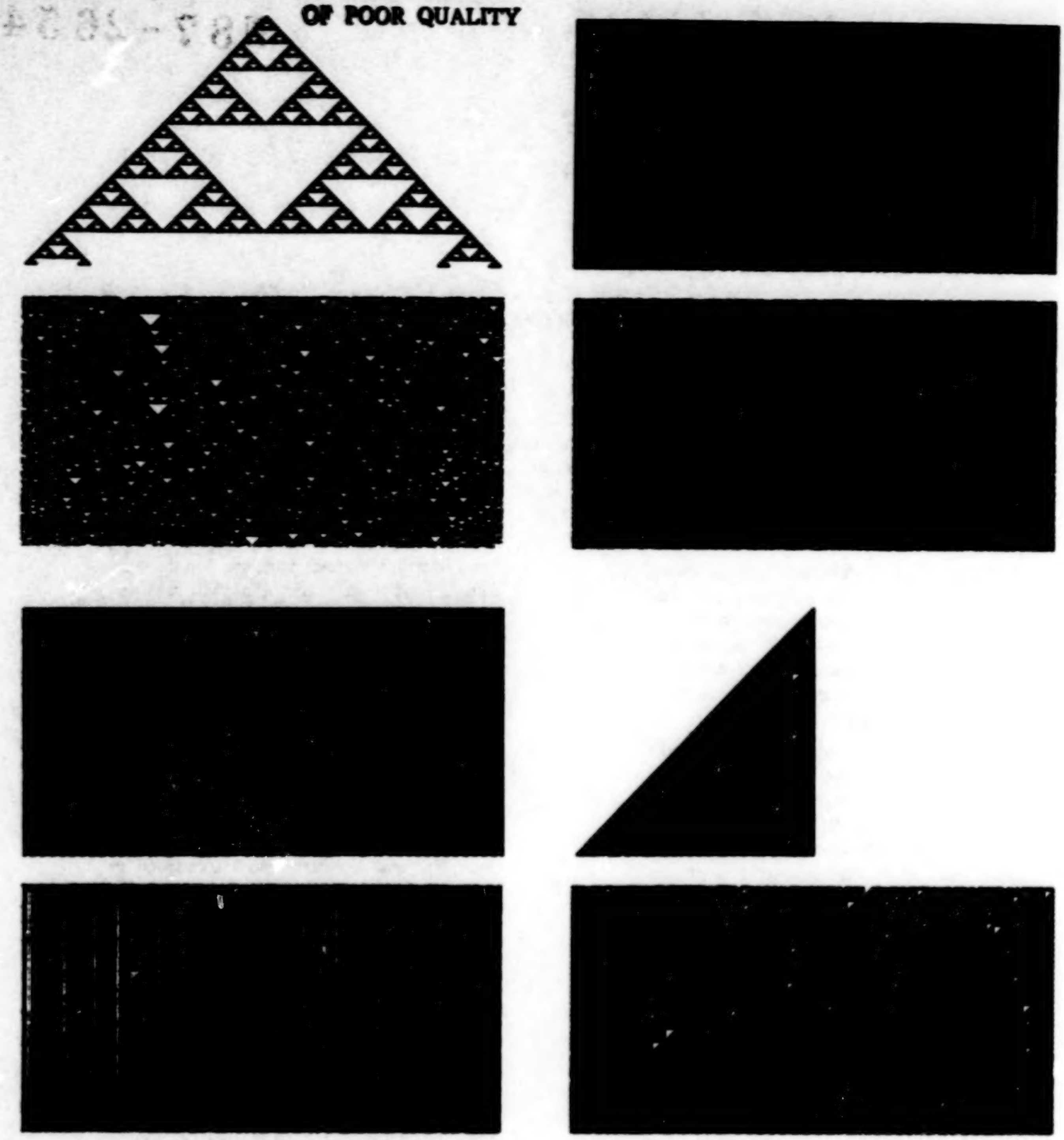


Figure 1: Examples of patterns generated by simple one-dimensional cellular automata. The cellular automaton consists of a row of about 600 sites, whose values evolve with time down the page according to simple logical rules. The value 0 or 1 of each site (represented by white or black) is determined from its own value, and the values of its two nearest neighbours on the step before. Patterns generated by four different rules are shown. In each case, the pattern obtained with an initial state containing a single nonzero site is shown above, and a pattern generated with a random initial state is shown below. (In the notation of ref. [3], the rules are numbers 18, 45, 73 and 110.) Despite the simplicity of these cellular automata, the patterns generated show considerable complexity.

in this system by considering the values of bulk quantities such as particle density or momentum density, averaged over a large lattice regions.

Figure 2 shows some results obtained in this way. Detailed studies have demonstrated that many of the phenomena seen in actual fluid experiments can accurately be reproduced by this simple cellular automaton model. Figure 2 shows calculations of two-dimensional flow past a cylinder. The standard transition from steady flow to a regular vortex street is observed. Then at higher Reynolds numbers (dimensionless fluid flow rates) the vortex street is seen to become aperiodic, corresponding to the onset of turbulent behaviour.

The cellular automaton method used in figure 2 may well be practical for many fluid dynamics computations. Through its close correspondence with the underlying physics of fluids, it is straightforward to include many physical effects and constraints. Thus for example solid objects with arbitrary shapes, and possibly, say, flexible boundaries, can easily be treated. In our current implementation on a Connection Machine computer with 65536 processors, lattices of size say 4096×8192 can be updated at a rate of about 10^9 sites per second, allowing the fluid flow patterns around objects to be found interactively up to Reynolds numbers of several hundred. The readily scalable architecture of the Connection Machine computer makes much larger simulations with the same method quite feasible in the future.

At a theoretical level, cellular automaton fluid models can be analyzed by much the same methods of statistical mechanics as have been used in trying to derive the Navier-Stokes equations for physical fluids from the microscopic dynamics of real molecules. One approach is to use kinetic theory to derive transport equations for the average densities of particles with particular positions and directions (e.g. [8]). In the hydrodynamic limit, these microscopic average densities can be approximated through a Chapman-Enskog expansion in terms of macroscopic fluid densities and velocities. The resulting equations for these macroscopic quantities correspond closely with the usual Navier-Stokes equations. Just like a real fluid, however, the cellular automaton model contains definite higher-order corrections, not included in the Navier-Stokes equations. In addition, analytical methods provide only approximate values for parameters such as viscosity; accurate values must be obtained from explicit computer simulations.

A fundamental assumption of the kinetic theory method is that the microscopic configurations of particles can be specified purely in terms of probabilities, which are in turn determined by the values of averaged quantities. This is essentially equivalent to the assumption of thermodynamic equilibrium, and is related to the fundamental principles of thermodynamics.

The Second Law of thermodynamics suggests that even if the initial configuration of particles is orderly, it will become progressively more disordered as a result of the motion and collisions of particles, and will show for example an increasing coarse-grained entropy. This phenomenon occurs if the evolution of the cellular automaton, even from "simple" initial conditions, yields behaviour that is so complicated as to seem random for practical purposes.

Very simple examples of cellular automata are known in which such apparent randomness can be produced. Figure 3 shows a one-dimensional example [9,10]. Even starting from an initial state containing a single nonzero site, many features of the pattern produced, such as the sequence of values in the center vertical column, are sufficiently random that they pass standard statistical tests of randomness [9]. The cellular automaton evolution thus acts like a pseudorandom number generator: even though a simple seed is given, the algorithm yields sequences whose simple origins cannot be discerned. The evolution of the system thus effectively "encrypts" the initial data: given just the output sequence it is very difficult to deduce the original seed. The cellular automaton of figure 3 can in fact be used as an efficient practical random sequence generator or stream encryption algorithm [11] (it is for example the primary pseudorandom generator used on the Connection Machine computer).

There are many mathematical systems which act in this way. It is for example easy to specify π, or to generate its digits. Yet once generated, the sequence of digits seems random for all practical purposes. Observations of this kind are related to the general conjecture of computational complexity theory (e.g. [12]) that $P \neq NP$. Computations which can be performed in polynomial time (P) seem to have inverses (which must be in the class NP) that require

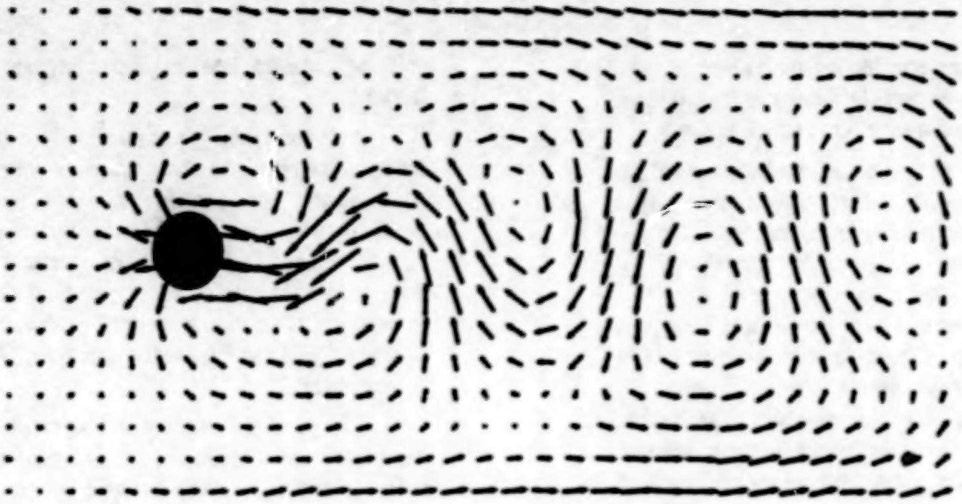


Figure 2: Fluid flow pattern obtained from a simple two-dimensional cellular automaton, simulated on a Connection Machine computer. The cellular automaton consists of 4096×2048 site hexagonal grid. Each site carries up to six discrete particles, which move and collide according to a simple discrete idealization of molecular dynamics. On a small scale, the particle motions appear random. But on a large scale, there is evidence that their average motion corresponds to that expected from a fluid which obeys the usual Navier-Stokes partial differential equations. In this figure, particles are injected on the left, leading to a net fluid motion from left to right. A circular obstacle is inserted in the fluid, and the resulting fluid velocities are computed by averaging individual particle velocities over 96×96 site regions. The velocities in the figure are shown transformed to the frame in which the obstacle is moving, and distant fluid is at rest. The simulation corresponds to a dimensionless Reynolds number around 100, and shows the formation of a "vortex street" behind the cylinder, as observed in physical experiments. The computations were performed with help from Bruce Nemnich and Jim Salem, on a Connection Machine computer with 65536 Boolean processors. The results shown were obtained after 10^5 time steps.

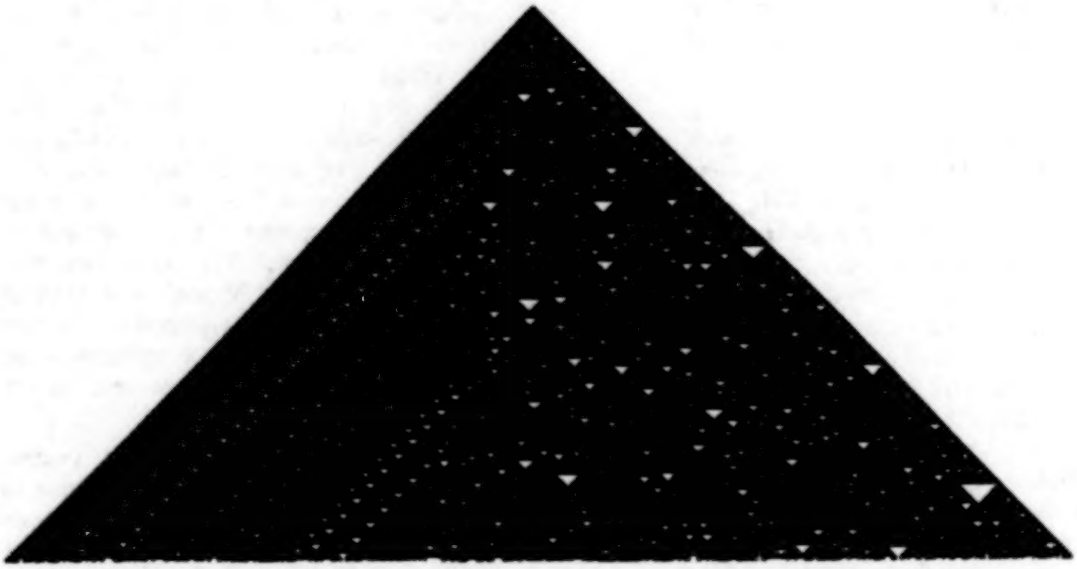


Figure 3: Pattern generated by a one-dimensional cellular automaton with two possible values at each site, and rule $a'_i = (a_{i-1} + a_i + a_{i+1} + a_i a_{i+1})$, starting from a single nonzero site. Despite the simplicity of its specification, many aspects of the pattern seem random. For example, the center column of site values passes all standard statistical tests of randomness. This cellular automaton illustrates the rather general phenomenon that simple processes can lead to complexity that is so great that many aspects of it seem random.

equations to be made. Thus, for example, the performance of computers is often measured in terms of the rate at which they can carry out the floating point operations needed. In many cases, there seem to be limitations which will prevent rapid increases in such performance.

Significant progress may perhaps more easily be made by somewhat shifting the emphasis. The kinds of operations which can efficiently be carried out by digital electronic circuits, and thus digital computers, are quite clear. Large numbers of simple logical operations can be performed, potentially in parallel on many elements of a regular grid. Given the structure, one may then ask the question of whether accurate computational models based on this structure can be found for physical and other systems.

Cellular automata (e.g. [3,4]) provide one class of examples. A cellular automaton consists of a discrete lattice of sites. Each site carries a discrete value, chosen from a small set of possibilities. The values are updated in a sequence of discrete time steps, according to logical rules which depend on the values of neighbouring sites. Cellular automata are thus, by construction, almost ideal for simulation on digital electronic computers. They are particularly well suited for the coming generation of massively-parallel machines, such as the Connection Machine computer [5], in which a very large number (currently 65536) of separate processors, each simple, act in parallel.

One of the most remarkable results of recent studies on cellular automata is that even with very simple rules, it is possible to obtain behaviour of considerable complexity [3,4]. Figure 1 shows a few examples. The rules consist of just a few simple logical operations. But when they are applied over and over again, their collective effect can yield very complex patterns of behaviour. Often these show striking similarities to forms seen in many natural systems, and in other mathematical models for these systems. Chaotic behaviour, corresponding to strange attractors, is common in cellular automata. Fractal patterns are also, for example, often produced.

One thus expects that very simple computational models, based say on cellular automata, should suffice to reproduce many different natural phenomena. The challenge is to abstract the essential mathematical features of the phenomena, so as to be able to capture them in as simple a model as possible.

As one example, I shall discuss here some recent models for fluid flow phenomena, based on cellular automata (e.g. [6]).

Fluids are conventionally described by the Navier-Stokes partial differential equations (e.g. [7]). These equations can presumably in principle describe the important phenomenon of fluid turbulence. But digital computer simulations based on the Navier-Stokes equations are barely able to reach the regime needed to reproduce turbulence accurately. Of course, the Navier-Stokes equations are themselves an approximation. At a fundamental level, fluids consist of discrete particles, usually molecules. The Navier-Stokes equations give an approximate continuum description of the average behaviour of large numbers of such discrete particles. When the Navier-Stokes are simulated on digital computers, however, discrete approximations must again be made. These approximations, perhaps in the form of finite differences, bear little resemblance to the original system of discrete particles. Yet in the limit of a large number of discrete elements, they too should correspond to the continuum Navier-Stokes equations.

A wide variety of systems, with very different microscopic dynamics, in fact appear to follow the Navier-Stokes equations in the large-scale limit. Thus, for example, air and water, despite their very different molecular constitution, can both be described by the Navier-Stokes equations, albeit with different values of parameters such as viscosity.

In an attempt to devise the most efficient computational models for fluids, one may try to find the simplest microscopic dynamics which reproduces the Navier-Stokes equations in the macroscopic limit. Such models may correspond to optimal algorithms for determining the behaviour of a fluid using a digital computer.

One class of computational models is based on a simple discrete idealization of molecular dynamics [6]. Particles move in discrete steps along the links of a fixed lattice, with each link supporting say at most one particle. The particles collide and scatter according to simple logical rules. The rules are arranged so as to conserve the total number of particles, and the total momentum carried by these particles. Fluid behaviour can potentially be obtained more than polynomial time, and probably often correspond to computations that are infeasible in practice.

Many mathematical models of physical processes probably show such behaviour [10]. Even with simple initial data, they rapidly yield configurations which seem random for practical purposes. Such behaviour may well be the basis for the widespread validity of the Second Law of thermodynamics. One of its important consequences is that a probabilistic or statistical description should indeed be valid for many systems such as cellular automaton fluid models. Such a description would depend only on macroscopic average variables. This may explain why different microscopic models often yield the same macroscopic behaviour. It is basic reason that simple discrete dynamics can give essentially the same overall behaviour as the full dynamics of physical molecules.

Statistical descriptions of cellular automaton fluid models are close in form to explicit finite difference approximations to partial differential equations. In both cases, each site on a grid carries a continuous variable which describes the average density and velocity of the fluid at that point. In practical computations with the finite difference method, this variable is typically represented directly as a floating-point number. In the cellular automaton method, the variable can be viewed as represented in a probabilistic or statistical fashion.

Following the usual development of statistical mechanics, a statistical description of a cellular automaton fluid can be obtained as an average over an ensemble of possible microscopic particle configurations. But an actual cellular automaton fluid simulation involves the evolution of just a single, specific, microscopic configuration. Nevertheless, following a fundamental assumption of statistical mechanics, one expects that suitable space or time averages of this specific configuration should yield results which are close to those obtained from averages over the whole ensemble.

This interpretation allows a comparison between cellular automata and discrete approximations to partial differential equations. In the latter case, ensemble average properties are considered, and their evolution is followed precisely. In the former case, just a single instance of the ensemble is considered, and macroscopic quantities are obtained as explicit averages over microscopic variables. If the fundamental assumptions of statistical mechanics are indeed valid, one expects that the cellular automaton method cannot fail to be more efficient than the finite difference one. For much of the information manipulated in the finite difference case is undoubtedly irrelevant to the macroscopic behaviour of interest.

Some evidence for this comes from the fact that most fluid computations yield results which are accurate to at most the percent level. Yet in the finite difference approach, fluid velocities at individual grid points are typically stored to 16-decimal-digit accuracy. Presumably it is only the most significant few digits, and certain overall features of less significant digits, which affect the final results. In the cellular automaton method, all bits of information about microscopic particle configurations are equally important. The cellular automaton representation may thus be a more efficient encoding of the state of the fluid.

The cellular automaton approach to fluid dynamics is but one example of an expanding set of computational models which are based on the collective properties of large numbers of simple discrete components. Standard cellular automata with deterministic rules have been used as models for reaction-diffusion systems, dendritic growth processes, dynamic spin systems, aggregation processes, and many other phenomena (e.g. [3]). Intrinsically probabilistic rules can also be used, and their consequences deduced by Monte Carlo sampling. The resulting models have been used extensively in studying quantum fields and many other systems.

In practice the probabilistic elements of such models must be implemented on digital computers using pseudorandom number generation algorithms. The resulting complete computational model, including the pseudorandom number generator, must thus be entirely deterministic. And since even very simple deterministic cellular automata can yield a high degree of randomness, one expects that formally probabilistic models can be replaced by deterministic ones, often involving a smaller total number of steps. One example of this occurs for the Ising spin system model, which is conventionally studied by updating spins probabilistically, but for which a more efficient algorithm based on a simple deterministic cellular automaton is known [13].

In general, there may be many different cellular automaton models for any particular system. Although the microscopic rules are different, their large-scale or continuum behaviour may be equivalent. In seeking the most efficient simulation algorithm for a particular system, one must find the "simplest" cellular automaton rules which yield the required large-scale behaviour.

Most computational models are created by explicit construction. Like most computer programs, each step or feature of their construction is specifically designed to have particular, known, consequences. But in most cases, this methodology will not yield truly optimal programs. Instead, one may imagine defining particular goals or constraints, and then searching the space of possible programs for the optimal ones which achieve these goals (e.g. [14,15]). This approach is particularly promising for problems such as finding optimal cellular automaton rules, in which the space of possible programs has a comparatively simple structure. Thus for example one may consider searching for the simplest cellular automaton rule which has a particular form of large-scale behaviour. Typically the space of possible rules can be reduced by imposing certain constraints, such as microscopic conservation laws, but the suitability of any particular rule can usually be determined essentially only by explicit simulation. The randomness-generating rule of figure 3 was found by such a search-based method.

The problem of finding optimal cellular automaton rules is in many ways analogous to problems such as the optimization of Boolean logic circuits, or the layout of large-scale integrated circuits. The overall goal is defined by the function to be implemented, but the most efficient circuits or rules can usually not be obtained by explicit construction. Instead one searches a large number of candidates, typically using a computer, and finds which of them is best.

Rather than performing an exhaustive search of possible circuits or rules, it is often better to use an iterative or adaptive procedure. One begins with a particular circuit or rule which has been constructed to satisfy the constraints that have been imposed. Then one makes a sequence of "moves" in the space of possible rules or circuits, with each move arranged so that the constraints are still satisfied. In the simplest cases, each move is chosen to yield a circuit or rule which is more optimal, or may be considered to have a lower "cost". But such a "gradient descent" method can find optima only when the "landscape" associated with the problem (whose height gives the cost for a circuit represented by a particular point) is essentially a smooth bowl. For many actual problems, the landscape seems closer to a "mountainous" or fractal one, on which the gradient descent method will get stuck in local optima. Simulated annealing seems to be a more promising general technique for optimization in such cases [16]. With this method, randomness is introduced into the choice of moves. Initially, a high level of randomness is used, so that the moves are sensitive only to the gross features of the landscape. The randomness is progressively decreased, so that optimization is carried out with respect to smaller and smaller scale features of the landscape.

As one example of such "adpative programming", I have recently been searching for the simplest one-dimensional cellular automaton rule which reproduces the diffusion equation in the large scale limit. (For another example, see ref. [17]). The rule must conserve a scalar additive quantity (analogous to particle number), but must generate randomness on a microscopic scale. In addition, the rules were chosen to be microscopically reversible, so that, by analogous with real physical systems, the evolution of the system can be uniquely reversed. Figure 4 shows the behaviour of a rule found by a search over a particular class of simple rules. Starting from a simple initial state, the rule generates progressively more random microscopic configurations. Although the simple initial conditions can in principle be recovered at any time by reversing the evolution, it becomes progressively more difficult to do so. As discussed above, this phenomenon may well illustrate the fundamental basis for the Second Law of thermodynamics. With the rule of figure 4, macroscopic average densities should follow the diffusion equation. As a result, slow spatial variations in density are for example damped on average according to the diffusion equation.

This paper has discussed some new directions for computational modelling. The fundamental principle is that the models considered should be as suitable as possible for implementation on digital computers. It is then a matter of scientific analysis to determine whether such models can reproduce the behaviour seen in

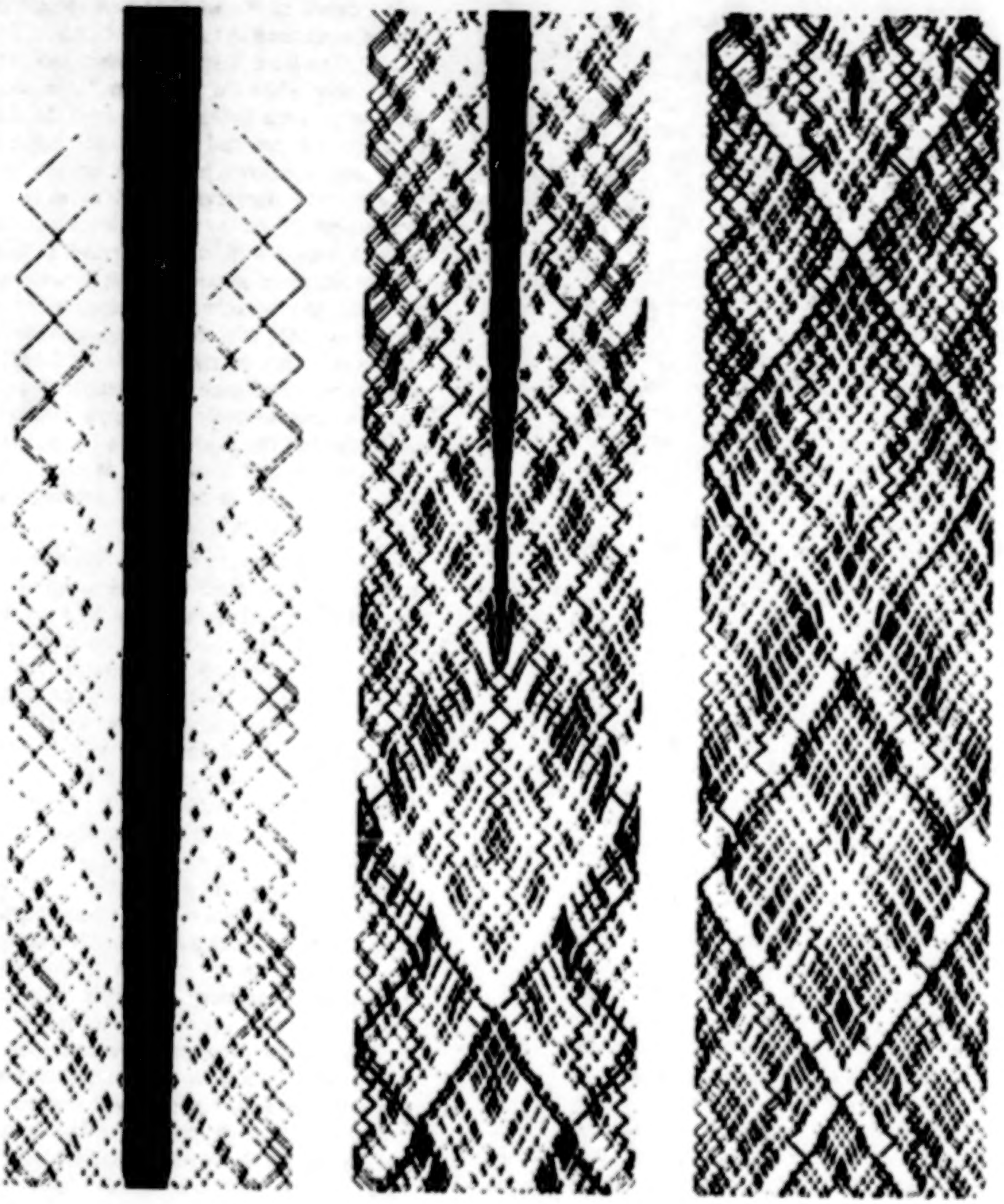


Figure 4: Pattern generated by a simple cellular automaton rule intended to mimic one-dimensional diffusion. Starting from a simple initial state, the reversible cellular automaton rule yields states that seem progressively more random. Such behaviour corresponds to that expected from the Second Law of thermodynamics, and can form the basis for simple discrete cellular automata to show macroscopic average behaviour which mimics continuum phenomena.

physical and other systems. Such analysis has now been carried out in several cases, and the results are very encouraging.

References

1. S. Wolfram, "Computer software in science and mathematics", Scientific American (September 1984).
2. A. Sommerfeld, *Partial differential equations of mathematical physics*, (Academic Press, 1955).
3. S. Wolfram (ed.), *Theory and applications of cellular automata*, (World Scientific, Singapore, 1986).
4. S. Wolfram, "Cellular automata as models of complexity", Nature 311 (1984) 419.
5. W. D. Hillis, *The Connection Machine*, (MIT Press, 1985).
6. U. Frisch, B. Hasslacher and Y. Pomeau, "Lattice gas automata for the Navier-Stokes equation", Physical Review Letters 56 (1986) 1505 [reprinted in ref. 2]; J. Salem and S. Wolfram, "Thermodynamics and hydrodynamics with cellular automata", in ref. 2; S. Wolfram, "Cellular automaton fluids 1: Basic theory", Journal of Statistical Physics, (to appear, November 1986); B. Nemnich and S. Wolfram, "Cellular automaton fluids 2: Basic phenomenology", Center for Complex Systems Research/Thinking Machines Corporation report, to appear.
7. D. J. Tritton, *Physical fluid dynamics*, (Van Nostrand, 1977).
8. A. Sommerfeld, *Thermodynamics and statistical mechanics*, (Academic Press, 1955).
9. S. Wolfram, "Random sequence generation by cellular automata", Advances in Applied Mathematics 7 (1986) 123.
10. S. Wolfram, "Origins of randomness in physical systems", Physical Review Letters 55 (1985) 449.
11. S. Wolfram, "Cryptography with cellular automata", in Proceedings of CRYPTO '85 (Santa Barbara, August 1985).
12. J. Hopcroft and J. Ullman, *Introduction to automata theory, languages, and computation", (Addison-Wesley, 1979).*
13. M. Creutz, "Deterministic Ising dynamics", Annals of Physics 167 (1986) 62; Y. Pomeau, "Invariant in cellular automata", Journal of Physics A17 (1984) L415.
14. D. Farmer, A. Lapedes, N. Packard and B. Wendroff (eds.), *Evolution, games and learning*, Physica D, to appear.
15. S. Wolfram, "Approaches to complexity engineering", in ref. 2, and to appear in ref. 14.
16. S. Kirkpatrick et al., "Global optimization by simulated annealing", Science 220 (1983) 671.
17. N. Packard, "Adaptation in the space of cellular automaton rules", Center for Complex Systems Research report, in preparation.

SIGNAL & IMAGE PROCESSING SESSION

Session Chair: Prof. Edward W. Davis
North Carolina State University

PRECEDING PAGE BLANK NOT FILMED

N87-26543

BLOCK ITERATIVE RESTORATION OF ASTRONOMICAL IMAGES WITH THE MASSIVELY PARALLEL PROCESSOR

Sara R. Heap
Laboratory for Astronomy and Solar Physics
Goddard Space Flight Center

Don J. Lindler
Advanced Computer Concepts
Potomac Maryland

ABSTRACT

We describe a method for algebraic image restoration capable of treating astronomical images. For a typical 500 x 500 image, direct algebraic restoration would require the solution of a 250,000 x 250,000 linear system. We use the block iterative approach to reduce the problem to solving 4900 121x121 linear sytems. We have implemented the algorithm on the Goddard Massively Parallel Processor, which can solve a 121 x 121 system in approximately 0.06 seconds. Here, we show examples of our results for various astronomical images.

Keywords: image restoration, constrained least-squares, block iteration

1. INTRODUCTION

The discrete model of linear image degradation is specified by the equation:

$$b = Hx + n \qquad (1)$$

where b and x are the pixel values of the degraded and original undegraded images stacked into column vectors, H is a matrix constructed from the impulse response (or point spread function) of the degradation, and n is an unknown additive noise vector. The object of restoration is to determine x, given b and possibly information on the properties of n. If the point spread function used to construct H is not known for the given optical-detector configuration, it must be estimated from the blurred image b. The point spread function is most easily estimated from point sources (i.e. stars) on the blurred image.

Since H may be ill-conditioned or singular, and only the statistical properties of the noise are known, there are many solutions, $\hat{x}$, for x which satisfy equation (1). In order to obtain a reasonable solution for x, it is necessary to utilize properties of x. The success of the restoration will therefore depend on the ability to model and apply to the restoration, known or assumed properties of the desired solution. Properties of x may consist of its "smoothness" or the restriction that all values in x be positive.

Some advantages of algebraic image restoration are:

1) The point spread function may be spatially variant;
2) If a constrained least squares method is used, the applied constraints may be varied from pixel to pixel to make maximum use of the known image properties;
3) Missing or bad pixel values in the blurred images can be easily handled without directly attempting to repair their values;
4) Noise properties can vary from pixel to pixel.

PRECEDING PAGE BLANK NOT FILMED

The main disadvantage of algebraic image restoration is the size of the linear system. For a 500 x 500 pixel image, H is a 250,000 x 250,000 matrix. Even with the most powerful computers available (including the MPP), a direct solution of the system would be impossible. In the next section, we describe a technique -- the block iterative method, of solving large linear systems.

2. THE BLOCK ITERATIVE RESTORATION ALGORITHM

2.1 Block Jacobi Iteration

In most astronomical images, the point spread function has a much smaller spatial extent than the image, so it is appropriate to work on the image locally. We therefore divide the image into blocks and restore each block separately, using values from the previous iteration as estimates of the unblurred image values outside the block. In most instances the blurred image is a good choice for the starting or zeroth iteration. This type of iteration is called block Jacobi or group Jacobi.(Young 1971) iteration and can be formulated in matrix notation as follows.

Consider the blurred image, b, divided into m blocks of equal size B_i, i = 1,m.

$$B=\begin{vmatrix} B_1 & B_2 & \cdots & & \\ & B_{i-1} & B_i & B_{i+1} & \\ & & \cdots & & \\ & & & B_{m-1} & B_m \end{vmatrix}$$

Stack the elements of each block and place them into a vector:

$$B=\begin{vmatrix} B_1 \\ B_2 \\ \cdots \\ B_m \end{vmatrix}.$$

Ignoring the noise for now, we write the system as:

$$HX = B ,$$

where H is partitioned into blocks:

$$H=\begin{vmatrix} H_{11} & H_{12} & \cdots & H_{1m} \\ H_{21} & H_{22} & \cdots & H_{2m} \\ & \cdots & & \\ H_{m1} & H_{m2} & \cdots & H_{mm} \end{vmatrix},$$

and X contains the restored values, blocked in the same manner as B. If the image were divided into blocks of n pixels each, then the blocks H_{ij} would have size n x n. The block Jacobi method can now be written as:

$$H_{ii}X_i^{r+1} = B_i - \sum_{\substack{j=1 \\ j\neq i}}^{m} H_{ij}\, X_j^r \quad , \qquad (2)$$

i=1,...,m, and where X_i^r is the stacked values for iteration r of block i. If we define the vector on the right hand side of equation (2) as $BMOD_i$ (i.e. the blurred image less contributions from outside the block as estimated from the previous iteration of the undegraded image), the linear system for block i can now be written as:

$$H_{ii}\, X_i^{r+1} = BMOD_i \quad . \qquad (3)$$

Using the block Jacobi method, we can reduce the problem to solving m smaller systems of size n x n of the form:

$$H\ x = b \qquad (4)$$

where H is H_{ii} for block i; x is X_i^{r+1} for block i, iteration r+1; and b is $BMOD_i$ for block i.

The solution for block i now requires the solution of an n x n linear system. For example, to restore a 100 x 100 pixel image divided into m=100 blocks, each of size, nxn=10x10, the largest system to be solved would have H_{ii} of size 100 x 100. Since solutions of linear systems require on the order of n^3 operations, the block approach compares favorably to the direct solution of the 10,000 x 10,000 system. For a spatially invariant point spread function, the problem is further reduced since H_{ii} will be identical for all i=1,m.

If a constrained least squares approach is used to solve the linear system, the solution will converge to acceptable results even with a block size as small as the full-width-at-half- maximum (FWHM) of the point spread function. Overlapping the blocks (accepting only the central portion for the next iteration) gives faster convergence or may produce convergence when no overlap results in divergence.

2.2 Image Constraints

The block Jacobi method reduces the restoration to solution of many smaller linear systems, but it does not address the ill-conditioned nature of H or the presence of noise in the blurred image. An ill-conditioned matrix means small changes in b, caused by noise, yield large changes in the solution $x = H^{-1} b$. In this section, we show how constrained solutions can handle these problems.

In most images, the data vary smoothly except at isolated points or edges. For example, an image of a star field will vary smoothly, except at locations of individual stars. We can make use of this image property, smoothness, by applying a constrained least squares fit. Specifically, we minimize a linear operator $||Qx||$ (i.e. The sum of the squares in Qx), where Q is a matrix designed to control smoothness or other characteristics of the image (Twomey 1963, Philips 1962). For example, we can control smoothness in the one dimensional case by minimizing the second difference in the solution subject to some other constraint. If the statistical properties of the noise are known, we could minimize the second difference such that the norm of $||Hx-b|| = ||n||$; that is to say, the difference of the blurred image and the solution reconvolved with the point spread function should have the same properties as the noise. In this case (minimize the second difference), Q would have the form:

$$Q = \left| \begin{array}{cccccccc} 0 & 0 & & & & & & \\ -1 & 2 & -1 & & & & & \\ & -1 & 2 & -1 & 0 & & & \\ & & \cdots & & & & & \\ & & & & & -1 & 2 & -1 \\ & & & & & & 0 & 0 \end{array} \right|$$

We use the method of Lagrangian multipliers, sometimes called the method of undetermined multipliers, to make the constrained least-squares fit. The solution of x is then given by (Andrews 1977):

$$x = (H^T H + \gamma Q^T Q)^{-1} H^T b$$

where γ is the reciprocal Lagrangian multiplier. The value of γ can be iteratively selected to control the amount of constraint in the solution. The solution using Lagrangian multipliers place no restrictions on the form of Q. This flexibility allows the development of a variety of constraints depending on the known properties of the image.

Figure 1 shows the application of this constrained least squares filter for a test case with different values of γ_2. The subscript 2 is used to indicate that the constraint is the minimum second difference. Note in figure 1.C-1.E that as γ_2 increases, noise in the solution decreases, but "ringing" at the edges increases. The ringing results from an inappropriate constraint at edges: the second difference should have a large value at an edge and should *not* be minimized. We therefore minimize the second difference at every location *except* the edges by setting the rows of Q corresponding to the edge locations to zeros. Figure 1.F shows a restoration of the same test image when the second difference constraint is not applied at the edges. A significant improvement results.

A direct extension of the method to two dimensional images is to minimize the Laplacian at each point. The Laplacian operator has a value at each pixel equal to four times the pixel value minus the values of the four immediate neighboring pixels. We use the subscript, *7*, to indicate the presence of the Laplacian constraint. As before, we set rows of the matrix Q to zero when the Laplacian constraint is not appropriate (i.e. edges or point sources).

The constraint need not be binary: we can vary the amount of constraint between no constraint to full constraint for any pixel, simply by multiplying the appropriate row in Q by a constant factor running from 0 to 1.

Another useful constraint is to minimize the difference of x from a trial solution (i.e. minimize $||p-x||$). The solution using Lagrangian multipliers is given by (Twomey 1963):

$$x = (H^T H + \gamma_t I)^{-1} (H^T b + \gamma_t p)$$

where p is the trial solution, I is the identity matrix, and γ_t is the reciprocal Lagrangian multiplier. The subscript, *t*, will be used to identify

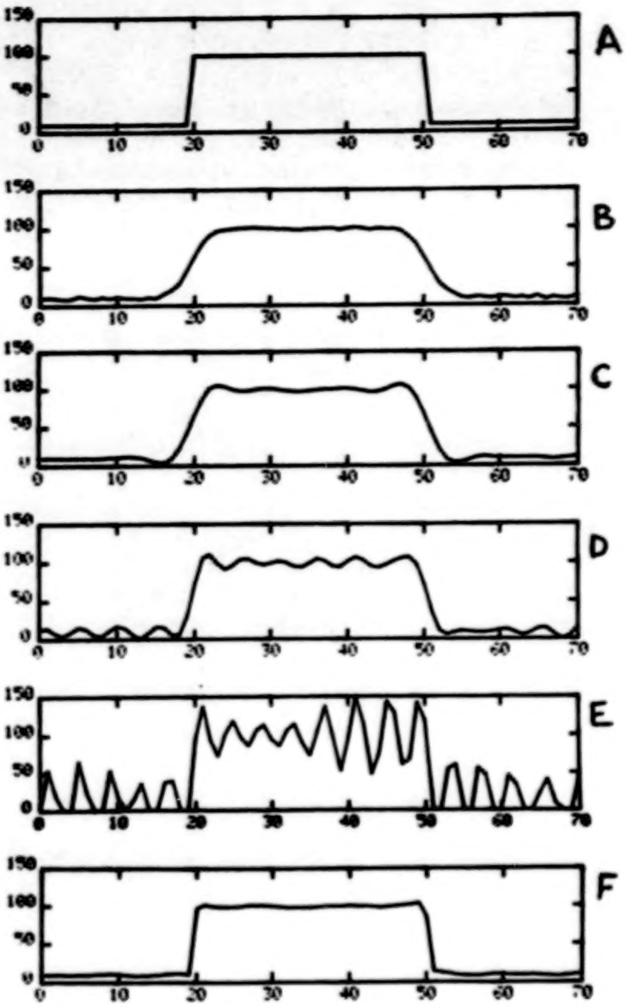


Figure 1. Effect of Langrangian multipliers. (A) original image; (B) image blurred with a Gaussian PSF (σ=2.0 pixels) and noise added (σ=1 DN); (C) restoration with γ_2=0.1; (D) restoration with γ_2=0.001; (E) restoration with γ_2=0.0001; (F) restoration with γ_2=0.1 with constraint removed at the two edges.

the constraint as minimization of the solution from a trial solution. Some possible choices for the trial solution, p, are a constant value (i.e. all zeros) or the blurred image itself. In either case, the ill-conditioned nature of H can be avoided and reasonable solutions obtained.

Multiple image constraints can be applied simultaneously:

$$x = (H^T H + \gamma_a Q_a^T Q_a + \gamma_b Q_b^T Q_b + \ldots + \gamma_t I)^{-1} (H^T b + \gamma_t p)$$

where a different value of γ can be selected for each constraint.

Selection of the reciprocal Lagragian multipliers is done by trial and error with the evaluation of γ by visual inspection of the results for various values of γ or by examination of the difference of blurred image and the solution reconvoled with the point spread function. This difference should have the same properties as the noise.

2.3 Missing or Bad Data Values

A problem results when trying to restore images with missing or bad data values (i.e. cosmic ray hits or bad CCD columns). If they are not taken into account in the restoration, the bad values will propagate to a larger portion of the output solution. To some extent, every point in the solution depends on all other values in the blurred image.

One method of handling bad pixels is to attempt to repair them before restoration by interpolating from neighboring values. This approach is successful only if the repair is accurate. An alternative method is to make no attempt at prior repair but handle them in the restoration process. In this approach, the restored image will have more data values than the blurred image, and the linear system is underdetermined and, therefore, singular (i.e. no direct inverse exists). To ignore bad data values, we set their corresponding rows in matrix H to zeros. This method of implementation (as opposed to removing row H creating a non-square underdeterminimed system) allows us to keep the matrix H square and decrease the complexity of implementation. Keeping H square in no way alleviates the problem of singularity. However, using the constrained least squares solution, the problem of singularity can be alleviated and reasonable solutions obtained.

3. IMPLEMENTATION OF THE ALGORITHM

The procedure for block iterative restoration described in section 2 is actually carried out over three computers. We use our laboratory VAX 750 with Gould DeAnza image display for interactive analysis of the blurred and restored images. We then use a local area network to copy the blurred image and point-spread function from the laboratory computer over to the MPP VAX 780 host computer. We use the latter machine to prepare input to the MPP, invoke the MPP, and reconstruct the output from the MPP into restored images. Preparation mainly involves dividing the various images (blurred image, constraint image, and trial solution) into blocks of 11x11 pixels, stacking them into vectors, and formatting them for access by the FORTRAN driver. Reconstruction of MPP output is the reverse procedure. The MPP itself is saved for the computer-intensive tasks of matrix inversion and matrix multiplication.

The primary software system that we use for interactive image analysis is Interactive Data Language (IDL, Research Systems Inc., Denver CO). To generate a single PSF from the intensity distributions of stars on the blurred image, we use DAOPHOT by Peter Stetson at Dominion Astrophysical Observatory.

We have also installed IDL on the MPP VAX-host as the user's high-level language to guide the restoration. The following IDL statements constitute the complete set of commands to restore an image stored as the variable, BLUR, with a point-spread function, PSF, and a block-size of 11 and step-size of 7 (i.e. the blocks overlap, and only the central 7x7 portion of a block is retained). C is an image controlling the constraint for each pixel, varying from 0.0 (no constraint) to 1.0 (full constraint) for each pixel. The two reciprocal Lagrangian multipliers, γ_l and γ_t, are both set to 0.001. TRIAL is the trial solution, and X is the first estimate of the restored image. The routine, dnext, invokes the next iteration. The last statement reads in the output from the MPP stored in the file, 'OUT.TMP', into the variable, X.

```
IDL> setblur,BLUR,PSF,11,7
IDL> setgamma, .001, .001
IDL> setcon, C
IDL> settrial, TRIAL

IDL> dnext, X
----   MPP   ----
IDL> bresult, X
```

Transfer of data and computations are carried out by the FORTRAN driver on the VAX 780 and Parallel Pascal and assembly language on the MPP. The appendix gives the PP code for matrix inversion and matrix multiplication implemented on the MPP.

Typically one iteration takes a few minutes of CPU-time when the VAX/MPP is not bogged down with other users. This time does not include wait-time for the MPP, overhead in transferring the data, etc. Since it is possible to examine an image *while* the MPP task invoked by DNEXT is running, the turn-around time is short enough that interactive work is a reasonable proposition. For restoration, it is essential: the eye can spot minute but systematic imperfections in the restored image as well as catch glaring errors.

The matrix multiplication algorithm given above takes 0.03 seconds to multiply two 128x128 matrices, and 0.06 seconds to invert the same size matrix. A typical image (512x512 pixels) requires the solution of about 5000 linear systems of size 121x121. Consequently, one iteration (block size of 11x11 pixels, step size of 7 pixels in both line and sample directions, spatially varying constraints) takes about 11 CPU-minutes. On a VAX 11/750, the identical procedure would take over 3 cpu-days!

4. APPLICATION OF THE ALGORITHM

We now describe the practical application of the algorithm, by considering four types of astronomical images requiring restoration.

Case 1: Clusters of point sources. Examples are double stars or crowded star fields, such as globular clusters. For the brighter sources, specialized observational techniques, such as speckle interferometry, are obviously superior approaches to higher resolution. For fainter sources in crowded fields, such as Cepheids in other galaxies, these techniques may not be feasible, and deconvolution of the image data at hand may be necessary.

Case 2: Point-source juxtaposed to or superposed on an extended source, where the point source is much *brighter* than the underlying extended source. The extreme example of this case is the quasar/host-galaxy, in which the nucleus of the galaxy (the quasar) has a surface brightness $\sim$10 times that of the underlying galaxy. Even the far wings of the quasar's point-spread function swamp the light from the galaxy, making it difficult to ascertain what kind of galaxy plays host to a quasar. Deconvolution of a quasar image holds the promise of separating, in effect, the quasar from the galaxy so that the galaxy can be examined directly. If the galaxy has structure, such as spiral arms, deconvolution will enhance the structural detail.

Case 3: Point-source juxtaposed to or superposed on an extended source, where the point-source is much *fainter* than the underlying extended source. This situation occurs in "deep sky" images, where the peak core brightness of a galaxy may be only 1% of the underlying sky. Astronomical seeing conditions or some other blurring mechanism may push the "nose" of a faint galaxy below the detection threshold. Conversely, deconvolution of the image may extend the detection limit.

Case 4: An extended source with structure. There are numerous examples of this case, such as planetary atmospheres or galaxies with dust lanes or jets. Usually, the structure is too arbitrary or complex to deduce the detailed physical structure via the route of convolving the point-spread function with some analytical model. Direct deconvolution is necessary.

In the following section, we show examples of each of these cases, paying particular attention to selection of the constraint factors, which as we noted earlier, are often a matter of judgement.

5. EXAMPLES

5.1 *Detection and photometry of objects in a crowded field* (image courtesy, T. Stecher, GSFC).

The UV rocket image of galaxy M101 (figure 2, upper left) suffers from a motion blur due to instabilities in the rocket's pointing system. Most sources in the image are OB/HII associations, which can be considered point sources at this resolution. It is of astronomical interest to make photometric measurements of these HII regions.

Figure 2 (upper right) shows the restoration using a constant constraint, γ_l=0.01 and γ_t=0.01, for all pixels in the image. The blurred image served as the zeroth iteration and as the trial solution. Some increased resolution is evident. We then used a simple thresholding technique to locate the HII regions in the first iteration (upper right image) and made a second iteration with the constraints removed at the locations of the detected HII regions. Also in the second iteration, we increased γ_t by a factor of 10 (for reasons explained in the next paragraph). As shown in figure 2 (lower left) the HII regions are better resolved, so that fluxes of individual HII regions can now be estimated.

Figure 2. Ultraviolet Image of M101.
Upper Left: Original, Blurred Picture;
Upper Right: First Iteration (with constant constraints);
Lower Left: Second Iteration (with variable constraints).

Figure 3. Computer-Generated Star Field. *Left:* Original, Blurred Image; *Right:* Restored Image.

We used a computer-generated star field (figure 3) to investigate the accuracy of detection and photometry that can be achieved in a restored image. The left image shows a blurred star field with a point spread function equal to the sum of two Gaussians, one with $\sigma = 2.0$ pixels, the other with $\sigma = 4.0$ pixels. The latter has a peak flux of 10% of the first. Three hundred stars were generated at random positions and with random magnitudes. Magnitudes ranged from 443 peak counts for the dimmest star to 90,750 peak counts for the brightest star. In an attempt to make the simulation realistic, we constructed each star individually using an analytical form of the point spread function, so that star centers fall at arbitrary positions between pixel centers. We simulated counting-statistic noise by adding Gaussian- distributed random numbers scaled by the square root of the counts in each pixel. In the first iteration, we used constant constraints for all pixels, with γ_l is set to 0.001. We experimented with different values of γ_t and found that setting γ_t at 0.01 gives the best results. In the second iteration (figure 3, right), we removed the constraints at the locations of stars detected on the first iteration.

A comparison between star detection in the blurred image and the restored image shows no significant improvement for the stars with a neighbor less than 3.5 pixels away. The most significant improvement in detection is for the 54 stars having neighbors between 3.5 to 5.0 pixels away (i.e. less then the full width-half maximum of the point spread function, which is approximately 5 pixels). Only 21 out of 54 stars are detected in the blurred image, while 48 stars are detected in the restored image. The average photometric error for these stars is 8%. For stars with separations greater than 5.0 pixels, the average error is 6%.

5.2 *Deconvolution of the quasar, 2130+099 (II Zw 136)* (courtesy, T. Heckman, U. Md.)

This quasar is a relatively bright ($m_V = 14.8$), low-redshift ($z = 0.06$) quasar. Figure 4 (top) compares the cross- sectional profile of the quasar with that of a nearby star on the same image. The difference between the two profiles is the contribution of the host galaxy. The aim of the restoration is to distinguish the host-galaxy from the quasar.

We found that it is essential to center the point-spread function *exactly* on the quasar (i.e. to a hundredth of a pixel); otherwise the misalignment causes ringing in the restored image. We used the intensity distribution of a nearby star to represent the point-spread function at the location of the quasar. To the extent that there is noise in this distribution or the PSF is spatially variant, the assumed PSF will be in error and generate spurious data in the restored image.

ORIGINAL PAGE IS
OF POOR QUALITY

As γ_1 increases, the noise in the solution decreases but ringing starts to set in. We need to impose the constraint for the host galaxy to avoid excess noise and to release the quasar from the Laplacian constraint to avoid the ringing. Thus, we could set the constraint image to 1.0 everywhere except at the location of the quasar and its immediate neighbors.

The host galaxy is not much brighter than the background sky, so we let the trial solution be given by the sky brightness (254 counts per pixel) everywhere except for at the quasar, where we set it to 3.5×10^5 (computed as the quotient of the maximum count-level in the quasar (21990) divided by the maximum count-level of the PSF (0.07).

The restored image should have properties somewhere between the original (blurred) image and the trial solution. Thus, we set the first estimate, X, of the restored images by calculating:

$$X = BLUR - T \otimes PSF + T \ ,$$

where the middle term at right is the convolution of the trial image and the point-spread function.

The cross-sectional profile of the restored image is shown in figure 4 (bottom). The quasar is now seen for what it is: a galaxy with an exceedingly bright nucleus. The morphological structure of the galaxy is consistent with its interpretation as a spiral galaxy.

The contrast level between the quasar and the host galaxy is more than 1000. Thus, "ringing" at the 1% level in the restored image would completely wipe out the galaxy image. This is why deconvolution of the quasar image is such a difficult problem. Deconvolution in one dimension has been carried out before (Bendinelli et al. 1984), but a full 2D deconvolution such as this requires the MPP.

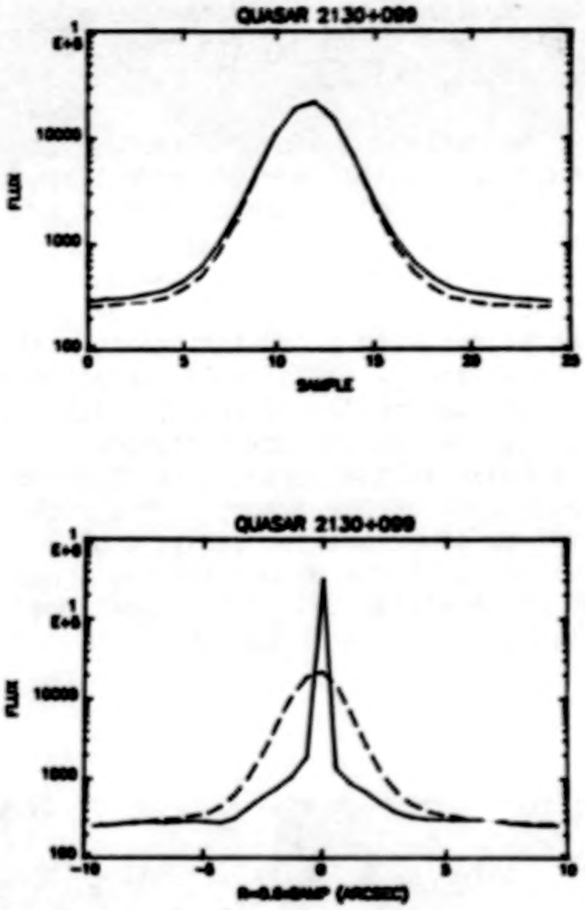


Figure 4. Cross-sectional profiles of the quasar, 2130+099. Top: profile of quasar (solid line) and nearby star (dashed line). Bottom: profile of restored image (solid line) and original, blurred image (dashed line).

5.3 *Restoration of Voyager images of Jupiter and Ganymede.* (courtesy, E. Danielson, CalTech)

The left-hand side of figure 5 shows images of Jupiter and Ganymede taken by Voyager. A point spread function was constructed using an image of a star taken with Voyager. The images on the right show the restoration after two iterations with a constant constraint with γ_l and γ_t set to 0.03 (selected by visual examination of results with various values for the reciprocal Lagragian multipliers). The improved resolution in the images on the right will be important for analysis of weather patterns on Jupiter and study of planetary detail with images from the Hubble Space Telescope.

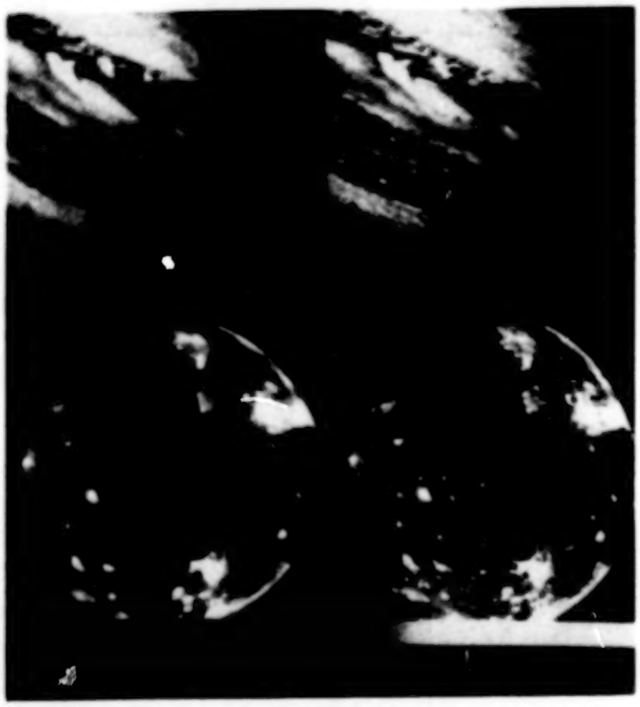


Figure 5. Voyager Images of Jupiter and Ganymede.
Left: Original, Blurred Images;
Right: Restored Images.

REFERENCES

Andrews, H. C., Hunt, B. R., *Digital Image Restoration*, (Prentice Hall: New Jersey), pp. 148-149 (1977)

Bendinelli, O., Lorenzutta, S., Parmeggiani, G., and Zavatti, F., "Deconvolution of photographic images of quasars from point spread functions", *Astron. Astrophys.* 128, 337-342 (1984).

Philips, D. L., "A technique for the numerical solution of certain integral equations of the first kind", *J. ACM*, 9, 84-97 (1962).

Smith, E., Heckman, T., Bothun, G., Romanishin, W., Balick, B., "On the nature of QSO host galaxies", *Astrophys. J.* 306, 64-89 (1986).

Twomey, S. "On the numerical solution of the Fredholm integral equations of the first kind", *J. ACM* 10, 97-101 (1963).

Young, D. M., "Iterative solution of large linear systems", (Academic Press: New Yourk), pp. 434-437 (1971).

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX

A.1 Matrix Inversion

The matrix inversion implemented on the MPP in Parallel Pascal uses Gaussian elimination with no pivoting. The following Parallel Pascal code will invert the matrix A, size M + 1, where X, COLK, and ROWK are 128 x 128 floating point arrays. ROWFLAG and COLFLAG are 128 x 128 boolean arrays. ROW-INDEX and COL-INDEX are 128 x 128 integer arrays which contain the row index and column index of each element respectively (indices run from 0 to 127). Initially, X contains the matrix A to be inverted and upon completion X will contain B, the inverse of A. Columns in X will be referred to by columns in matrix A or B, whichever is appropriate since both matrices are stored in X.

```
1. for K:= 0 to M do
2.  begin
3.   ROWFLAG:= ROW-INDEX = K;
4.   COLFLAG:= COL-INDEX = K;
5.   rowbroad(X,COLK,128,COLFLAG);
6.   where COLFLAG do X:= 0;
7.   where COLFLAG and ROWFLAG do X:= 1;
8.   where ROWFLAG do X:= X/COLK;
9.   colbroad(X,ROWK,128,ROWFLAG);
10.   where not ROWFLAG do X:=
            X-COLK*ROWK;
11.  end;
```

Line 1 begins a loop on each column K in the matrix X. Line 3 sets the boolean variable ROWFLAG true for elements in row K and false everywhere else. Line 4 sets COLFLAG true for elements in column K. ROWFLAG and COLFLAG will be used as masks for subsequent operations. Line 5 takes the Kth column in matrix A and propagates each element along its corresponding row. We no longer need to retain the Kth column of matrix A. This column can now be used to store the Kth column in matrix B, which initially is set to 1 in row K and zero elsewhere. This is done by lines 6 and 7 which set column K to all zeros (line 6) and then set b_{kk} to 1 (line 7). Line 8 divides row K by the value in position a_{kk}. (Remember that line 5 had propagated the value of a_{kk} along the entire row K). Line 9 will take the Kth row and propagate each element along its corresponding column. Line 10 is the real work horse and subtracts a_{ik} * row k from every row except k in A. Since B is also stored in X, the same operations are automatically performed on B. This step would have produced zeros in column k of the original matrix A in every location except a_{kk} which would have the value of one. Upon completion of the loop for all columns, the matrix X will contain the inverse.

A.2 Matrix Multiplication

The matrix product of two arbitrary size square matrices A and B can be written as:

$$C = \quad (A_i \otimes B_i \quad)$$

where n is the number of rows and columns in the matrices, and $\otimes$ indicates element by element multiplication (not matrix multiplication). The following Parallel Pascal code gives the implementation of the algorithm on the Massively Parallel Processor. A and B are the input matrices and C will contain the result. M is the number of rows and columns minus one. COLFLAG and ROWFLAG are boolean variables and AI and BI are matrices used to store the propagated columns and rows of A and B. ROW-INDEX and COL-INDEX are defined in Section A.1.

```
C:= 0;
for I:= 0,M do
 begin
  ROWFLAG:= ROW-INDEX = I;
  COLFLAG:= COL-INDEX = I;
  rowbroad(A,AI,128,COLFLAG);
  colbroad(B,BI,128,ROWFLAG);
  C:= C + AI * BI
 end;
```

N87-26544

RECONSTRUCTION OF CODED APERTURE IMAGES

Michael J. Bielefeld
Astronomy Programs, Computer Sciences Corporation
Space Telescope Science Institute
3700 San Martin Dr. Baltimore, MD 21218

Lo I Yin
Code 682, Solar Physics Branch
Laboratory for Astronomy and Solar Physics
NASA Goddard Space Flight Center, Greenbelt, MD 20771

ABSTRACT

Balanced correlation method and the Maximum Entropy Method (MEM) were implemented to reconstruct a laboratory x-ray source as imaged by a Uniformly Redundant Array (URA) system. Although the MEM method has important physical advantages over the balanced correlation method, it is computationally time consuming because of the iterative nature of its solution. MPP, with its parallel array structure is ideally suited for such computations. These preliminary results indicate that it is possible to use the MEM method in future coded-aperture experiments with the help of the MPP.

INTRODUCTION

In the energy range of hard x-rays, i.e., 30 keV to 100 keV, there is no focusing optics. Consequently, in order to image the many interesting objects which emit radiations in this energy range such as galaxies, the sun, nuclear reactor components, and human organs tagged with radioisotopes, one must resort to the use of collimators or pinholes made from high density materials. In this paper, we will focus on the use of pinholes.

Because a single pinhole is extremely inefficient, there is a strong interest in the use of multiple pinholes to image x-ray objects [1-12]. However, in most cases, the images on the detector formed by the many pinholes strongly overlap each other resulting in a detected image that is not recognizable. A decoding process must then be applied to the detected image in order to recover or reconstruct the image of the original object. Such multiple-pinhole masks are usually referred to as coded apertures.

In principle, the large collection efficiency (close to 50%) of coded apertures offer the possiblility of a greatly enhanced signal-to-noise ratio while maintaining the high spatial resolution of a single pinhole. Furthermore, for terrestrial applications, coded apertures can provide tomographic information of the x-ray object. In practice, however, there are

PRECEDING PAGE BLANK NOT FILMED

several difficulties associated with the use of coded apertures, especially in the imaging of extended or large x-ray objects. Because of the strongly overlapping images on the detector, the signal in one location of the reconstructed image may contain contributions from all other portions of the object. This type of signal cross-talk is object-dependent and can be present in addition to statistical noise. Such signal-cross talk can cause severe contrast degradation in the reconstructed image of extended objects. There has been many studies in recent years on the performance properties of various kinds of coded apertures. One of the most promising type of coded apertures is the Uniformly Redundant Array (URA) [10,11]. A URA is a special kind of multiple-pinhole mask in which the number of times a particular separation occurs between any pair of pinholes is the same for all separations. The separations are therefore uniformly redundant. URA has some very desirable properties; one of which is that with proper decoding the signal cross-talk mentioned above can be eliminated completely. However, noise cross-talk still exists. That is, the statistical noise from one part of the object can still contribute to the signal of another part in the reconstructed image.

A few years ago, we proposed another simple alternative: the Non-Overlapping Redundant Array (NORA) [12]. It consists of a regular array of pinholes (e.g., a hexagonal array) where the separation between pinholes as well as the separation between the NORA mask and the detector can be carefully chosen such that the images on the detector formed by the individual pinholes do not overlap. We have shown that in NORA, there is neither signal cross-talk nor noise cross-talk in the reconstructed image and the only inherent noise in the system is that due to counting statistics. The signal-to-noise ratio of NORA, assuming Poisson statistics, is always square-root of N times that of a single-pinhole camera, where N is the total number of pinholes. This is true even for extended objects and is the ideal limit achievable by a multiple-pinhole system. Another important feature of NORA is that it is possible to reconstruct the extended x-ray object in 3-D by simple optical correlation. We have already demonstrated in the laboratory that the optically reconstructed image can be viewed in true 3-D with both horizontal and vertical parallax. In addition, NORA should also provide quantitative tomographic information through digital reconstruciton. It is this latter goal that prompted us to seek out the capabilities of the Massively Parallel Processor (MPP).

DIGITAL DECODING

General

Although it is possible to reconstruct an image by analog method when NORA is used as the coded aperture, digital reconstruction is mandatory when the coded aperture, such as URA, produces overlapping images on the detector. Furthermore, to obtain quantitative tomographic information, digital computation is always necessary.

In digital decoding, the large number of pinholes, pixels and mathematical operations demand large amounts of computing time even with available fast algorithms. At present, because we are still investigating various coding and decoding methods which involve many repeated trials and iterative calculations, long computation times and turn around delays can be both costly and frustrating. The MPP, with its parallel array structure is ideally suited for this type of computations. In fact, as we will show below, the MPP makes our investigations feasible, while the conventional mainframe computer, in normal use, has proven to be inadequate.

Decoding Methods for URA

Balanced Correlation Method - Because of the uniformly redundant and the cyclic nature of the URA, its point spread function is a delta function with constant and flat sidelobes. That is, if the object is a point source and is detected by an ideal URA system, the decoded image by means of an autocorrelation operation will be also a point source (delta function), but with a constant and uniform background. This flat dc background can be eliminated by using the balanced correlation method. In this method, although the decoding array has the same pattern as the coding URA array with 1's representing the holes, the non-holes are represented by -1's rather than 0's [10]. However, in contrast to well-separated point sources, noise due to statistical fluctuations in the background which is not aperture related can still contribute to the reconstructed signal as the object gets large, even with balanced correlation decoding. This kind of noise cross- talk may give rise to artifacts in the low-contrast background region of the reconstructed image.

Maximum Entropy Method (MEM) - Recently many investigators have become interested in applying the maximum entropy method (MEM) to the field of image restoration including the reconstruction of coded-aperture images [13-18]. MEM is an iterative method which maximizes the configurational entropy while using prior knowledge such as chi-squared (X^2) statistic and total detected intensity as constraints. Through iteration, the solution with the maximum configurational entropy, i.e., with the least configurational information, is selected from a set of solutions all of which satisfy the chi-squared fit of the actual data. This solution is considered as the most likely estimate of the original object that is consistent with the available data.

Following Willingale [15], the solution has the form:

$$\hat{f}_i^m = z_i e^{-(\mu+1)} \cdot e^{-2\lambda\left[\sum_K B_{Ki}(\hat{d}_K - d_K)/\sigma_K^2\right]} \quad (1)$$

where ($\hat{f}_i^m$) is the estimated intensity of the ith pixel of the object as seen by the instrument which maximizes the configurational entropy [$-\sum \hat{f}_i \ln \hat{f}_i + \sum \hat{f}_i \ln z_i$]; (z_i) is the instrument efficiency, i.e. (f_i / z_i) is the true intensity of the ith pixel of the object; (B_{ki}) is the transpose of the blurring matrix of the coded aperture; (d_k) is the actual data on the detector; ($\hat{d}_k$) is an estimated data, without noise, which would be produced on the detector if the object were correctly represented by ($\hat{f}_i$); (σ_k^2) is the variance of the data (d_k); (λ) and (μ) are Lagrange multipliers. The function Q which is being maximized to produce the solution as represented by Equation (1) is:

$$Q = -\sum_i \hat{f}_i \ln \hat{f}_i + \sum_i \hat{f}_i \ln z_i - \lambda \sum_i (\hat{d}_i - d_i)^2 / \sigma_i^2 + \mu \sum_i \hat{f}_i \qquad (2)$$

The first two terms of (2) comprise the configurational entropy; the third term with the Lagrange multiplier (λ) is the (X^2), and the fourth term with Lagrange multiplier (μ) is the total intensity of the object. For large number of data points N, (X^2) $\cong$ N. To conserve total counts, ($\sum \hat{f}_i = \sum \hat{d}_i$). Noise in the data is accounted for by the variance (σ_k^2).

There are several important advantages in using (1) as the decoding solution. Because (1) is in the exponential form, this solution is never negative. The first exponential is a constant scaling factor which gives the reconstruction a uniformly distributed intensity without features. When the noise in the data is very high, this featureless solution (λ=0) will be consistent with the data, ($\hat{f}_i^m$) will be simply proportional to (z_i) by maximum configurational entropy. When the signal in the data is high, the featureless background as given by the first exponential will be modulated by the features provided by the second exponential. The summation in the second exponential represents a cross-correlation between the blurring function of the coded aperture and the difference between the estimated data and the actual data weighted by its statistical variance. Since this reconstruction occurs in the exponential, iterative algorithms are needed for its solution.

The relative weighting of entropy and (X^2) is controlled by (λ). As mentioned above, when (λ) = 0, (X^2) has no weighting, and the solution is a uniform distribution as given by maximum entropy. When (λ) is increased, the process reduces (X^2). A final (λ) will be selected when (X^2) becomes close to N, the expected value. To help convergence, we also adopted the search algorithm of Willingale [15] by taking weighted averages of successive iterations.

EXPERIMENTAL DATA

As an initial test toward digital decoding using the MPP we have chosen some data which we had obtained previously with a URA coded aperture. The experimental arrangement is sketched in Figure 1. The URA mask consisted of a two-cycle mosaic of a basic 15 x 17 m-sequence array. The pseudo-random m-sequence pattern was generated according to the procedure given by MacWilliams and Sloane [19]. The pattern was drilled into a 0.5-mm thick Pb sheet by a computer-driven lathe. The holes were 0.3 mm in diameter; the center-to-center separation of adjacent holes was 0.6 mm. Thus, the transparency of this mask was about 10%. For the imaging detector, we used a Lixiscope [20] with a digitizing anode. Briefly, for the present data, the Lixiscope consisted of a thin layer of $YSiO_3(Ge)$ powder serving as an x-ray to visible light converter which was deposited on the entrance faceplate of a 1:1 image intensifier containing a triple microchannel-plate (MCP) electron multiplier [21]. The output electron signals from the triple MCP are detected by a resistive anode which can provide both the position and the amplitude of an electron pulse. For the simplest case of the present experiment a single small I-125 x-ray source (28 keV) was used as the x-ray emitting object. The distance between the source, the mask and the Lixiscope were chosen such that the sensitive area of the detector recorded at least one complete basic array of the magnified shadow of the two-cycle URA mask. The experimental image of the source, which was positioned at 31 cm from the detector, is shown in the upper left corner of Figure 2. The display exaggerates the constrast in the data for this array of 256 x 256 pixels. The average counts per pixel is about 2. Because the emitting object is a point source, the basic URA pattern is clearly visible witin the circular active area of the detector. The digitized version of this image (Fig. 2) is used as the data to be decoded by both the balanced correlation method and the MEM.

DECODING OF URA IMAGE

The basic implementations of the balanced correlation method and the MEM are relatively straightforward. However, an important distinction should be mentioned between this type of x-ray image processing and that of the more common visible/IR image processing. In our case, one is dealing with extremely low count rates. Because of this, the statistical uncertainty of individual pixels has to be followed through the decoding process at the basic level of computation. The formalism for the MEM in Equation (1) takes full account of this requirement.

The specification of spatial resolution for the experimental system displayed in Figure 1 includes not only resolution in the x-y plane but also in the z direction. Hence, the digital decoding of 3-D objects requires much finer sampling than the basic pinhole array. This requirement for high sampling rates along with the iterative nature of the MEM are the main factors which directed us toward using the MPP.

The decoding process requires at a minimum sampling rate of 17 x 15 pixels per cycle of the URA. Because the detected image (Fig. 2, top left) is an array of 256 x 256 pixels, this image is collapsed to the minimum array of 17 x 15 through summing as shown in Figure 2, top right. This coded image is almost featureless. The bottom images of Figure 2 show the results of digital decoding by the MEM (left) and the balanced correlation method (right). Both methods clearly reconstructs the point source to the same degree. However, these images illustrate several advantages of the MEM. First, the MEM does not permit the physically impossible negative counts seen in the background region of the image reconstructed by the balanced correlation method (bottom right). Secondly, the MEM produces a much smoother background (bottom left). This smoothness helps to minimize the erroneous interpretation of artifacts arising from noise cross-talk.

As mentioned earlier, the iterative process should be terminated when (X^2) become close to N, the number of data points. Letting (X^2) to reduce further will only add artifacts to the already smooth background. (λ) controls the relationship between the entropy portion and the data portion in Equation (1).

For extended objects with unknown shapes, the smoothing capability of the MEM becomes even more important than for points sources. In our opinion, this capability of suppressing background artifacts amply justifies the computational expense of the MEM. Timing experiments on the MPP indicate that MEM decoding of our x-ray URA data is indeed feasible. In general, the MEM requires approximately ten times more computational power than the balanced correlation method because of the iterations. The CPU requirements for the basic filtering kernel calculation increases as the factor N for the MPP, but as the square of N for a typical mainframe computer, where N is the total number of pixels in a decoded data array. For a future experiment which requires the reconstruction of five tomographic planes at ten iterations per plane with three values of (λ), our estimate is that decoding would take 1.5 minutes of MPP/CPU time for the minimum sampling rate of 17 x 15. At a very high rate of 170 x 150 pixels per URA cycle, it would take about 2 hours. A compromising sampling rate of 51 x 45 would take about 12 minutes. Even for this compromised level of decoding, we estimate that the equivalent processing on a large un-vectorized mainframe would take in excess of 24 hours of CPU time.

CONCLUSIONS

The results of our work to date have been encouraging. The continuation of this research would be greatly enhanced with the computational power of the MPP. We need trial-and-error experience to find the optimum decoding algorithms as experimental configurations are refined, and to determine the practical tomographic depth resolution for 3-D x-ray objects. Our near future plans include the decoding of x-ray data obtained with NORA aperture using the MPP.

ACKNOWLEDGMENT

We acknowledge the continued support of our investigations from within NASA, especially the

MPP Working Group. It is a pleasure to thank Judy Devaney of SAR Corp. and of the MPP supporting staff for her invaluable insights and assistance. We are also indebted to D. Vitagliano and V. Krueger for their technical assistance with the manuscript.

REFERENCES

1. J.G. Ables, Proc. Astron. Soc. Aust. 1, No. 4. 172 (1968).

2. R.H. Dicke, Astrophys. J. 153, L101, (1968).

3. H.H. Barrett and F.A. Horrigan, Appl. Opt. 12, 2686, (1973).

4. T.M. Palmieri, Astrophys. Space Sci. 26, 431, (1974); 28, 277 (1974).

5. C. Brown, J. Appl. Phys. 45, 1806, (1974).

6. B. MacDonald, L.T. Chang, V. Perez-Mendez, and L. Shiraishi, IEEE Trans. Nucl. Sci. NS-21, 672, (1974).

7. R.G. Simpson, H.H. Barrett, J.A. Suback, and H.D. Fisher, Opt. Eng. 14, 490, (1975).

8. F. Gunsen and B. Polychronopulos, Mon. Not. R. Astron. Soc. 177, 485, (1976).

9. L.T. Chang, B. MacDonald, and V. Perez-Mendez, Proc. Soc. Photo-Opt. Instrum. Eng. (SPIE) 89, 9 (1976).

10. E.E. Fenimore and T.M. Cannon, Appl. Opt. 17, 337, (1978); 18, 1052, (1979).

11. E.E. Fenimore, Appl. Opt. 17, 3562, (1978).

12. L. Yin, J. Trombka, S. Seltzer, and M. Bielefeld, Appl. Opt. 22, 2155 (1983).

13. S.F. Gull and G.J. Daniell, Nature 272, 686, (1978).

14. M. Sims, M.J.L. Turner, and R. Willingale, Space Sci. Instrum. 5, 109, (1980).

15. R. Willingale, Mon. Not. R. Astr. Soc. 194, 359, (1981).

16. S.F. Burch, S.F. Gull, and J. Skilling, Computer Vision, Graphics, and Image Processing 23, 113, (1983).

17. R. Willingale, M.R. Sims, and M.J.L. Turner, Nucl. Instrum. & Methods in Phys. Res. 221, 60, (1984).

18. S.F. Gull and T.J. Newton, Appl. Opt. 25, 156, (1986).

19. F.J. MacWilliams and N.J.A. Sloane, Proc. IEEE 64, 1715, (1976).

20. L. Yin, J. Trombka, and S. Seltzer, Nucl. Instrum. & Methods 158, 175, (1979).

21. L. Yin, J. Trombka, R. Schmadebeck, S. Seltzer, and M. Bielefeld, SPIE 268, 97, (1981).

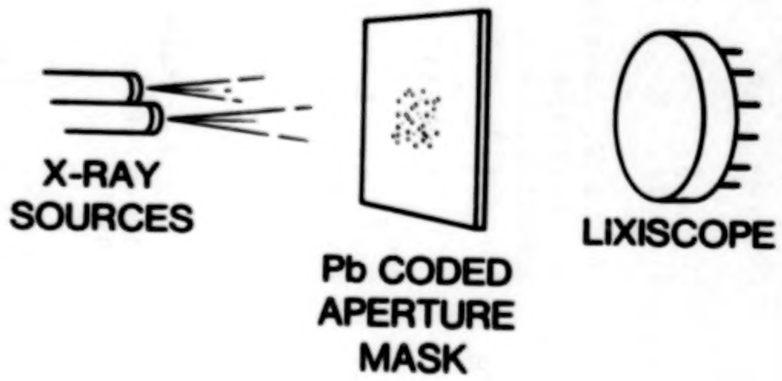

X-RAY
SOURCES
Pb CODED
APERTURE
MASK
LIXISCOPE

Figure 1. Sketch of experimental setup.

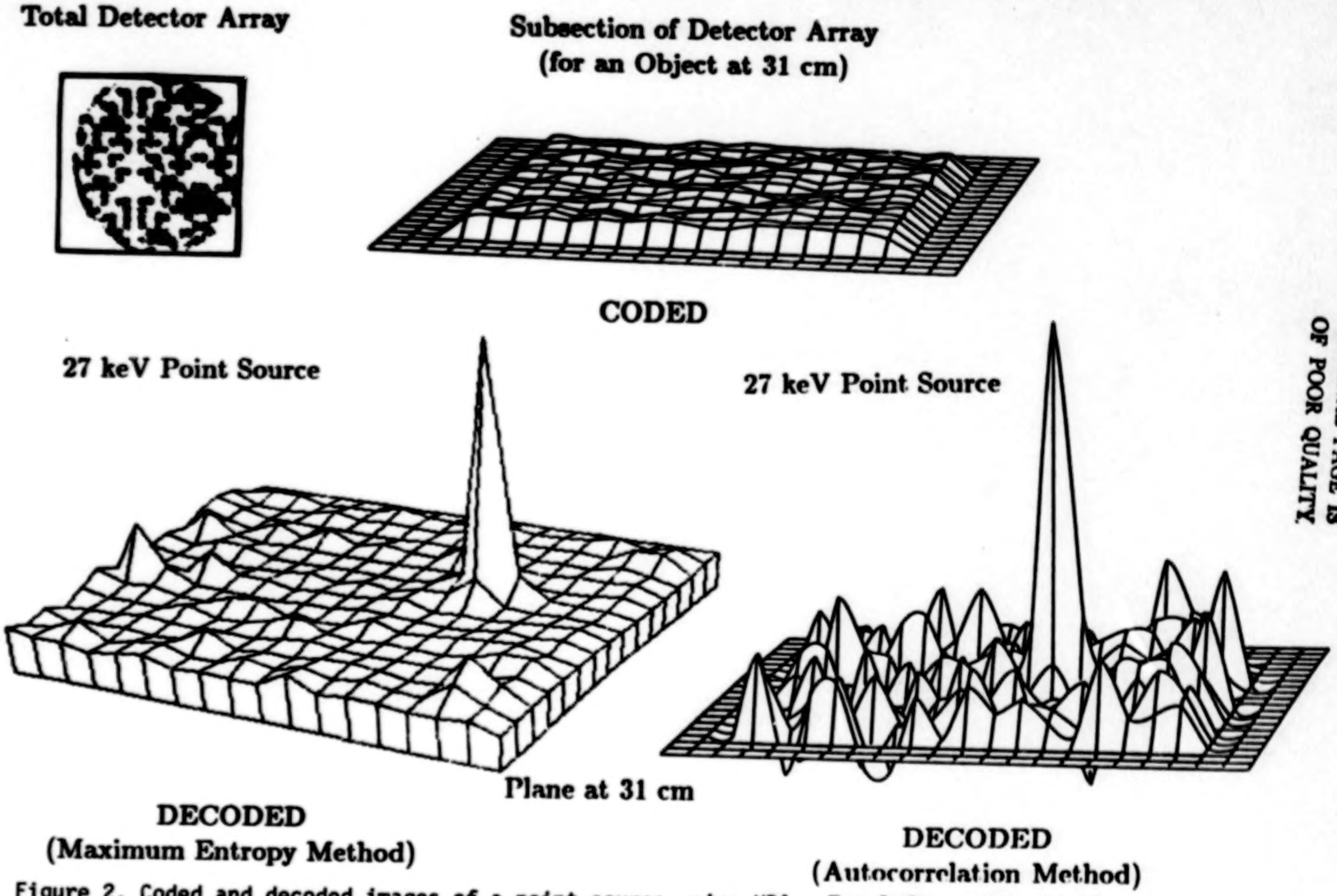


Figure 2. Coded and decoded images of a point source using URA. Top left: original data. Top right: digitized data. Bottom: decoded images by two different methods.

ORIGINAL PAGE IS
OF POOR QUALITY

SYNTHETIC APERTURE RADAR SIGNAL PROCESSING ON THE MPP

N87-26545

H. K. Ramapriyan
Space Data and Computing Division
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

E. J. Seiler
Science Applications Research, Inc.
Lanham, Maryland 20706

ABSTRACT

Satellite-borne Synthetic Aperture Radars (SAR) sense areas of several thousand square kilometers in few seconds and transmit phase-history signal data at the rate of several tens of megabits per second. For example, the Seasat SAR acquired data for a 100 km by 100 km area in about 18 seconds and transmitted at the rate of 85 Mbits/sec. The Shuttle Imaging Radar - B (SIR-B) has a variable swath of 20 to 50 km and acquired data over 100 kms along track in about 13 seconds. The transmission rates used were 30.4 and 45.6 Mbits/sec. The processing of the phase-history data into images with pixels representing the radar reflectances involves correlation with the two-dimensional point target response (i.e., the reference function) which is typically more than 1200 samples in each dimension. Even with the simplifying assumption of separability of the reference function, the processing requires considerable resources -- high-speed I/O, large memory and fast computation.

Processing systems with conventional hardware take several hours to process one Seasat image and about one hour for a SIR-B image. Bringing this processing time closer to acquisition times requires an "end-to-end" system solution. However, for the purposes of demonstration, we have implemented software on the present MPP configuration (with conventional I/O hardware on the VAX 11/780 host computer) for processing Seasat and SIR-B data. The software takes advantage of the high processing speed offered by the MPP, the large (32 Mbyte) Staging Buffer, and the high-speed I/O between the MPP array unit and the Staging Buffer. It is found that with unoptimized Parallel Pascal code, the processing time on the MPP for a 4096 x 4096 sample subset of signal data ranges between 18 and 30.2 seconds depending on options.

INTRODUCTION

Satellite-borne synthetic Aperture Radars (SAR's) sense areas of several thousand kilometers in a few seconds and transmit the phase-history signal data at the rate of several tens of megabits per second. The first space borne SAR was on the Seasat which was launched in June 1978. It acquired the data needed to generate a 100 km by 100 km image in about 18 seconds and transmitted at 85 Mbits/sec. The Shuttle Imaging Radar-B (SIR-B) flown aboard the Challenger during October 1984 with a selectable incidence angle capability had a variable swath of 20 to 50 km and acquired data needed for an image covering 100 km along track in about 13 seconds. The transmission rates used were 30.4 and

45.6 Mbits/sec. The space-borne radars planned for the future, with multiple frequencies, polarizations and incidence angles, will have similar or higher data rates. The processing of the phase-history data into images with pixels representing the radar reflectances involves correlation with a two-dimensional point target response (i.e., a reference function). In the case of Seasat SAR, this function extends over 1536 x 4200 samples. In the case of SIR-B, it is approximately 1200 x 1500 samples. Fortunately, it is possible to treat this two-dimensional correlation as a separable problem to a very close approximation and implement it as two one-dimensional correlations. Even so, processing the data in a reasonable time requires considerable resources: high-speed input-output, large memory and fast computation.

Processing algorithms for space-borne SAR's differ from those for air-borne SAR's due to the much larger range walk and range curvature corrections required for the former. Algorithms for space-borne SAR's have received considerable attention since 1978. Descriptions of processing algorithms can be found in papers by Cumming and Bennet (1979), Wu (1980) and Wu et al (1982). The execution of these algorithms on systems with conventional hardware (minicomputers, array processors, computer compatible tapes and disks) takes several hours for one Seasat image and about one hour for a SIR-B image. Bringing this processing time closer to acquisition times requires an "end-to-end" system solution. However, for the purposes of demonstration, we have implemented the processing algorithms on the present configuration of the Massively Parallel Processor (MPP) at the Goddard Space Flight Center (GSFC). A discussion of the MPP hardware and the host configuration system can be found in companion paper (Fischer, this volume). This implementation takes advantage of the high processing speed of the MPP, the large (32 Mbyte) Staging Buffer and the high-speed I/O between the MPP array unit and the Staging Buffer.

The purpose of this paper is to present the MPP implementation of the SAR processing algorithms and comments on the timings achieved.

2. MPP IMPLEMENTATION OF THE PROCESSING ALGORITHM

All signal processors must address, in different ways unique to the design goals and base processor hardware capabilities, the same basic processing functions, and compensate or correct for conditions such as range walk and range curvature. The basic steps in image generation can be categorized as preparation, signal processing and postprocessing as shown below:

Preparation

- Raw data extraction and reformatting
- Range Reference Function (RRF) generation
- Doppler parameter estimation for use in Range Walk Correction (RWC) and the Azimuth Reference Function (ARF)
- Frequency domain break-point calculation to facilitate range curvature correction
- Generation of trigonometric constants for use in the Fast Fourier Transform (FFT) algorithm.

Signal Processing

- Range compression
- Corner Turning (Transposition)
- Azimuth Compression
- Image Formation

Postprocessing

- Range Attenuation Compensation (RAC)
- Geometric Correction (conversion from slant range to ground range and rectification to a map projection).

The raw data extraction and reformatting consist of converting the received radar signal data from the collection medium (High Density Digital Tape - HDDT) to a medium acceptable to the processing system. Since the present configuration of the MPP and its host computer does not contain a reader for HDDT's, it is necessary to start processing on this system from a Computer Compatible Tape (CCT) or disk-file. Seasat and SIR-B signal data are available from Jet Propulsion Laboratory (JPL) on CCT's with records containing packed data. On these CCT's, Seasat data contain 4 bits per sample and the SIR-B data contain 3 to 6 bits per sample depending on the parameters selected for a given imaging interval (data take). For processing, the signal data records are unpacked into 4 or 8 bits per sample before reading into the staging buffer. The remaining preparation and signal processing steps are based on algorithms described by Wu (1982). The details of those algorithms are beyond the scope of this paper. However, the equations needed to maintain continuity of presentation and a discussion of the MPP implementation are given below. The post-processing steps of RAC and geometric correction are not considered here.

2.1 Preparation

2.1.1 Range Reference Function (RRF) Generation:

The RRF is the function used for range compression. It consists of a dechirp function (filter to match the transmitted chirp pulse) and a window function to reduce sidelobes of the matched filter response. The function is given by

$$c(t) = \cos 2\pi[(f_o + B/2)t - \phi(t)]$$

for $0 < t < T$
$= 0$ elsewhere

where f_o = range offset frequency (in Hz)

B = bandwidth of the transmitted pulse (in Hz)

$\phi(t) = 1/2\ Kt^2$

K = range frequency modulation (FM) rate (in Hz/sec)

$= B/T$

and T = pulsewidth (in seconds)

The values of f_o, B and T are shown below for Seasat and SIR-B.

	SIR-B	SEASAT
Range Offset Frequency, MHz	7.2	11.3825
Bandwidth, MHz	12.0	19.0
Pulsewidth, microsecs	30.4	33.8

Frequency domain weighting is used for sidelobe reduction. Several options exist for this. Hanning and Taylor window weighting have been used by JPL (personal communication, 1983) and Applied Physics Laboratory (APL) (Raff, 1983) respectively. A review of windowing and its effects on processed signals can be found in Harris (1978).

In our implementation we have used the Kaiser-Bessel window. This window provides a better combination of sidelobe suppression and compressed pulse width than the Hanning Window. If the window function is $B(\omega)$, then the RRF in the frequency domain is given by $C(\omega)^*B(\omega)$ where $C(\omega)^*$ is the complex conjugate of the Fourier transform of c(t).

To obtain the RRF in the discrete frequency domain, we embed the samples of c(t) in an N-element long complex array for some $N > T/f_s$ = signal sampling frequency), take its Discrete Fourier Transform (DFT), and multiply its complex conjugate by a sampled version of $B(\omega)$. The choice of N is a very important sizing consideration in the processor design. In our implementation, N=4096 for the reasons indicated below in subsection 2.2. In addition, for SIR-B, a notch filter is used to suppress the calibration tone. This is done by zeroing out the value of $C(\omega)^*B(\omega)$ for corresponding to the frequency of the calibration tone.

2.1.2 Doppler Parameter Computation:

The Doppler parameters are needed for computing the RWC and the ARF. Even though in our implementation the RWC occurs during range compression (and hence before azimuth compression), it is more convenient to define the ARF first and then discuss the parameters for RWC.

The ARF is given by (Wu, 1982)

$$h_1(x,r,r_o) = \sum_{i=1}^{n} g_i(x,r_o)\,\delta(r-d_i)$$

which is the range compressed response due to a point target located at $(0,r_o)$ where (x,r) are the (azimuth, range) coordinates at which h_1 is evaluated, $g_i(x,r_o)$ is the azimuth response function evaluated along a range bin at distance d_i from r_o, and is the Dirac delta function. Further,

$$g_i(x,r_o) = a_i(x) \exp\,[j\psi(x,r_o)]$$

where

$$a_i(x) = W_a(x)\ q\ (d_i - r_1(x))$$

$$\psi(x,r_o) = 4\pi r_1(x)/\lambda$$

$W_a(x)$ = Antenna weighting function in azimuth

$q(r)$ = range compressed point target response

$r_1(x)$ = Slant range from the satellite to the point target

= Wavelength of the transmitted electromagentic wave

Note that in the above equations, x=0 is the (arbitrary) origin corresponding to the time when the antenna beam center (in azimuth) passes the point target.

It is convenient to express $r_1(x)$ and $\Psi(x,r_0)$ in terms of the Doppler frequency and the Doppler rate induced by the relative motion between the spacecraft and the target. It is seen that when the relative acceleration is nearly constant,

$$\Psi(t) = \Psi(0) - 2\pi(f_d t + 1/2\ \dot{f}_d t^2)$$

where t is used as an independent variable proportional to x,

$$\Psi(0) = 4\pi r_0/\lambda$$

$$f_d = -\ 2\ R.V/\ (\lambda r_0)$$

$$\dot{f}_d = -2\ (r.A + V^2)/(\lambda r_0)$$

and R, V, and A are respectively the relative displacement, velocity and acceleration vectors of the target with respect to the spacecraft at t=0. (Note that r_0 is the magnitude of R).

An implementation consideration is to ensure that the number of range bins spanned by the range compressed data from a given target is kept to a minimum. This is accomplished by separating $\Psi(t)$ into two parts:

$$\Psi(t) = \Psi_1(t) + \Psi_2(t) \quad \text{where}$$

$$\Psi_1(t) = -2f_{d0}t$$

$$\text{and}\ \Psi_2(t) = \Psi(0) - 2[(f_d - f_{d0})\ t + 1/2\ \dot{f}_d t^2]$$

where f_{do} is a "nominal" doppler frequency (computed at the center of the image being processed). The function $\Psi_1(t)$ is merged into range compression processing to perform range walk correction which is equivalent to shifting each iso-azimuth line by a number of range bins proportional to $f_{do}t$.

It is important to determine f_d and $\dot{f}_d$ accurately, since the quality of azimuth compressed data is critically dependent on them. The values of f_d and $\dot{f}_d$ depend on the ephemeris, attitude, antenna pointing direction and position of the target (and therefore on the particular range bin of interest).

When the ephemeris and attitude data are known accurately (assuming known antenna pointing direction with respect to the spacecraft body axes), f_d and $\dot{f}_d$ can be computed from them.

When the ephemeris and attitude data are not known accurately, the signal data themselves are used for computing f_d and $\dot{f}_d$. The computation of f_d uses a clutter lock procedure. The $\dot{f}_d$ values are computed using an autofocussing technique.

Clutter lock: The clutter lock algorithm is based on the principle that in regions not dominated by strong point targets, the averaged magnitude of the azimuth frequency response matches a shifted aperture gain function. The shift in the aperture gain function required to match the above frequency response is the doppler frequency f_d. Since f_d is a function of range, it is necessary to determine this shift for several sets of iso-range lines. In our implementation, group of 16 contiguous iso-range lines are used at intervals of 64 lines, starting from the nearest range-bin. The procedure used is as follows. The range offset frequency is first removed from the data. The data are transposed for convenient access of iso-range lines. Sets of 16 lines (4096 samples each) are then Fourier transformed and the averages of the Discrete Fourier Transform (DFT) magnitudes are found. This averaged DFT magnitude array is correlated with the aperture gain function $f(a)$ given by

$$f(a) = (\sin a/a)^2 \text{ where}$$

$$a = 2\pi k/4096 \text{ for } k = -2047, -2046, \ldots, 2048, k \neq 0$$

and $f(0)=1$.

Now, f_d is given by

$$f_d = \text{PRF } (1 + K/4096)$$

where PRF = Pulse Repetition Frequency

and K = sample number at which the correlation peak occurred (ranging from -2047 to 2048).

After f_d is computed for each selected set of 16 iso-range lines it is assigned to the mid-range value of that set. Then, a least-squares fit is performed using the f_d values for all the selected sets to obtain f_d as a function of range r:

$$f_d(r) = f_{d0} + mr \text{ , where m is the slope determined by the fit}$$

The mid-range value of f_d is used for range-walk correction during the range compression process.

Autofocus: The autofocus algorithm is based on the principle that if the f_d value is correct, the azimuth compression will result in the best focused point target response. Thus, if the f_d value is correct, each look of multiple-look processed azimuth compressed data will place image-features at the same location. However, a small error in f_d will result in shifts among the images from the multiple looks. By determining the shifts, the corrections to f_d can be computed. For this method to succeed, it is necessary to have an initial estimate of f_d which is close enough to the correct value to result in some recognizable features in the multi-look processed images. The initial f_d values are estimated using the ephemeris and attitude data. In our implementation, the two looks closest to the center of the azimuth reference function are used to produce two images. These two images are then correlated to determine the shift between them. The shift is used to correct f_d and the process is repeated. This procedure is used on several sections of the image at various range values and f_d is computed as a linear function of range through a least squares fit.

2.1.3. Frequency Domain Break-Point Calculation:

The purpose of this preparatory step is to precompute some of the parameters needed for manipulating frequency domain data during azimuth compression. Wu (1982) has shown that the Fourier transform of the desired signal after azimuth compression can be written as

$$\Omega(u,r_0) = \sum_{i=1}^{n} [S\,(u,d_i + r_0)\,A(u,r_0)]\,H(u,r_0)$$

where

u is the azimuth frequency variable

r_0 is the range bin of interest

n is the number of range bins over which the range compressed point target response is spread

d_i is the distance from r_0 to the i^{th} range bin used in range curvature correction

S is the Fourier transform of the range compressed signal and

A_i and H are, respectively, the complex conjugates of the Fourier transforms of $a_i\ (x)$ and $\exp[j\psi(x,r_0)]$ defined in subsection 2.1.2.

Two commonly used approaches to approximating the summation in the above equation are nearest neighbor assignment and cubic interpolation. Conceptually, these approximations are treated as follows.

Let i_0 be the (possibly fractional) value of i for which $A_i\ (u,r_0)$ is a maximum. Note that i is a function of (u,r_0). Let I be the integer nearest to i_0. Let I_0 be the largest integer less than or equal to i_0. Then, the nearest neighbor assignment is equivalent to replacing $A_i\ (u,r_0)$ by $\delta(i-I)$. Cubic interpolation is performed using the samples of S corresponding to i in the range I_0-1 through I_0+2. (That is, the A_i have suitable "weight" values for i in this range and are zero elsewhere).

Since the range curvature depends only on f_d, the values of i_0 for the various u can be found from f_d and f_d. Therefore, for each set of f_d, f_d indicated in the avoce subsection, we can precompute i_0 as a function of u. For nearest neighbor assignment, which is our present implementation, it is most convenient to store I in terms of the values of u at which it changes. These are called the frequency domain break-points. These break-points are computed during this preparatory step and are used for generating the "masks" required for range curvature correction (see subsection 2.2.3).

2.2 SIGNAL PROCESSING

The signal processing primarily involves the computation of correlations between the recorded signal and the reference functions in the range and azimuth directions. These correlations are performed most efficiently as multiplications of the functions after a transformation to the frequency domain, followed by an inverse transformation to return to the time domain. Since this is the most computationally intensive portion of the processing, it is important to use a Fast Fourier Transform (FFT) algorithm which takes full advantage of the capabilities of the MPP.

The FFT design has been chosen to maximize its efficiency subject to the MPP hardware constraints such as the 1024-bit array memory per processing element (PE) and the 32 Mbyte staging buffer capacity. One of the design constraints is that it should be possible to transpose the range compressed data within the 32 Mbyte staging buffer prior to azimuth compression so that the low-speed data transfers between the MPP and the disk (or tape) devices are minimized. This requires that the signal processing be carried out on blocks of raw data with MN samples where M and N, the numbers of samples in the range and azimuth directions, respectively are chosen such that the results of range compression for the entire block can be held in the staging buffer. To minimize the computation time, the trigonometric constants needed for the FFT are precomputed and stored in the array memory. To eliminate the need for reloading trigonometric constants between range and azimuth compression, it is necessary to choose M = N. Further, the number of correct output values produced after correlation processing is N - RFL + 1 where the RFL is the number of samples in the reference function. Therefore, it is desirable to use as large a value of N as possible. These considerations lead to the selection of N = 4096.

In general, algorithms on the MPP are most efficient when all the PE's are kept busy performing "useful work" and the data transfers among PE's are minimized. (Note that the data transfer time can be comparable to computation time. Shifting a K-bit operand by x PE's requires (x+2)K clock cycles, while the addition of two K-bit operands requires 3K cycles). In the case of the FFT, the maximum efficiency is achieved when all the PE's are performing complex multiplications or additions and there is no data shuffling among the PE's. This requires that all of the data in an array to be transformed be resident in the memory of a PE. (In that case, 16,384 FFT's could be performed simultaneously). However, due to the limitation of 1024 bits of memory per PE, it is necessary to distribute each of the arrays to be transformed over several PE's. If each of the arrays to be transformed has N samples and is distributed over a rectangular subset of PE's of dimensions A x B, it is necessary to store C = N/(A x B) samples per PE. It is easy to see that the number of shift operations needed per FFT is of the order of (A + B). Thus the data transfers are minimized if A + B is minimized and C is as large as possible. The value of C is limited by the available array memory. For a given C, the value of A + B is minimized by selecting A = B. For the case N = 4096, A = B = 32 and C = 4 are chosen to provide the optimum data arrangement. This allows 16 FFT's to be performed simultaneously on the MPP.

The MPP is most efficient when performing integer arithmetic; this speed decreases as the length of the operands increases. Another consideration is that the transfer rate of data between the MPP and the host system is severely limited by the available hardware configuration. Thus in order to minimize the computation and transfer times the size of the operands should be chosen as small as possible while preserving the accuracy of the computations. For the SAR processing, the raw signal input is 3 to 6 bits per sample, and the processed output image is generally desired as 8 bits per pixel. Experimentation has shown that 16-bit integer calculations are sufficient to preserve full accuracy at the intermediate stages of computation.

2.2.1 Range Compression

The range compression is performed by reading the raw signal data in groups of 16 lines, performing the FFT, multiplying by the precomputed RRF performing range walk correction by multiplying by the walk reference function, performing the inverse FFT, basebanding the data, and outputting the resultant complex data.

The range walk correction is designed to minimize the number of range bins over which a point target's response extends during azimuth processing. The walk reference function is computed as described in section 2.1.2, using the value of f_d at the middle range value of the image determined by the clutterlock processing. The multiplication by the walk reference function results in a shift of each successive range line by a (possibly fractional) number of range bins.

In most signal processors the range offset frequency f_o is eliminated in a basebanding operation that shifts frequency domain data down in frequency by f_o. On the MPP, basebanding is accomplished after restoring the data to the time domain by multiplying the data by a complex operand to shift it in frequency. This method, requiring a single complex multiplication per sample, is more efficient than the time consuming data shuffling that would be necessary to shift the data in the frequency domain.

The resulting data are subsampled to obtain 2048 complex samples per line. Then, all 16 lines of range compressed data are reduced to 16-bits per sample (8-bits real and 8-bits imaginary) and transferred to the staging buffer simultaneously.

2.2.2 Corner Turning

The corner turning is performed by writing the 4096 x 2048 complex 8-bit data to the MPP staging buffer, and reading the data into the array memory in transposed order. The transposition is performed under the direction of a Stager Control Block (SCB) file, constructed to direct the staging buffer to use row-major order for input and column-major order for output of the 2-dimensional array. The staging buffer has a capacity of 32 Mbytes and a theoretical I/O rate of 80 Mbytes/sec (overhead considerations which reduce the achievable rate are discussed in section 3). Previously the staging buffer had a 2 Mbyte capacity and an I/O rate of 20 Mbytes/sec. This smaller capacity made it necessary to perform the corner turning by writing partially transposed data to a disk file, read the data back to the staging buffer using random access I/O, and complete the transposition in the staging buffer. The disk I/O has a theoretical rate of 1.2 Mbytes/sec, while the achievable rate depends largely on the system load at the time of processing. Thus at least a 67 fold increase in speed is obtainable by using the current staging buffer configuration. Further, at present, the data transfers between the staging buffer and host files require intervention of the host. If the transfers between the array unit and the staging buffer can be performed without host intervention the speed increase becomes much greater.

2.2.3 Azimuth Compression

The azimuth compression is performed on the MPP on a group of data referred to as an Azimuth Processing Blocks (APB's), consisting of 32 lines by 4096 azimuth values per line of range compressed data. The azimuth compression produces (4097-ARFL) "correct" data values per line, where ARFL is the length of the Azimuth Reference Function. Therefore, for single-look processing the APBs are overlapped by (ARFL-1) azimuth values to process all azimuth values.

For each APB an ARF is computed in terms of f_d and $\dot{f}_d$ determined by the clutterlock and autofocus algorithms as discussed in section 2.1.2. After the ARF is computed its Fourier transform is computed and stored.

The range compressed data are read 16 lines (range bins) at a time and the Fourier transformed using the same data arrangement as was used during range compression. Prior to multiplication by the Fourier transformed ARF's it is necessary to perform range curvature correction to assure that all the data corresponding to a given slant range occur in the same range bin. To accomplish this the data are first rearranged so that the data from the 16 range bins corresponding to a given azimuth occur in the same PE. The frequency domain break-points computed as in section 2.1.3 are used to determine the number of range bins by which each sample is to be shifted. The shifts (movement of data within each PE by a known integral number of range bins) are accomplished using masked assignment operations. The data are then rearranged in the "FFT order", multiplied by the Fourier transforms of the ARF's and inverse transformed.

2.2.4 Image Formation

The inverse transformed data above are complex. The magnitudes of these data constitute the fully correlated image result. The image values so obtained are accumulated in the staging buffer and written to disk as blocks of (2049 - RRFL/2) lines with (4097 - ARFL) pixels per line where RRFL and ARFL are the lengths of the range and azimuth reference functions, respectively.

3. RESULTS AND CONCLUSION

An example of a SIR-B image generated using the above MPP implementation is shown in figure 1. This image covers a region surrounding Mount Shasta, California. It represents the results of processing 4 blocks of signal data, each with 4096 x 4096 samples. The blocks are overlapped by ARFL = 2031 samples in the azimuth direction. The signal data are first singlelook processed to form image data as discussed above in section 2.2. Successive pixels in the azimuth direction are then averaged in groups of four to reduce speckle and approximately match the slant range pixel size.

The times required for processing one 4096 x 4096 block are shown in the table below. Processing and I/O times are reported separately. For the processing, the times also include the speed improvement resulting from eliminating the scaling from the FFT routine, and the use of a new, more efficient code generator.

The I/O times compare the previous transposition requiring the use of an intermediate disk file to the method that performs the transposition in the staging buffer. In our first implementation, the intermediate file consisted of 16-bit data. However, it was determined experimentally that there was no significant loss of image quality if only the 8 most significant bits each of the real and imaginary parts of the range compressed data were written to the intermediate file. We have use the 8-bit intermediate data for the implementation with transposition in the staging buffer.

The processing times for the range and azimuth compression reflect the speed advantage obtainable from the MPP, while it can be seen that the limitation of the system is in the I/O transfer rates. There are two major limitations to the current system that are reflected in the reported rates. The first is that data transfers from disk files are limited to approximately 1.2 Mbytes/sec. The second is that the I/O transfers require the intervention of the host computer, and the overhead involved in this intervention severely limits the rate of transfer. A transfer rate of 80 Mbytes/sec. is theoretically attainable for moving data between the array unit and the staging buffer. In the table are reported the estimated times for a conservative 20 Mbyte/sec. rate. Note that the use of the staging buffer reduces the transposition time to 1.4 seconds, which is almost negligible.

The processing times shown can be improved by optimization of the code. The FFT includes a routine that checks the range of data values at intermediate stages of the computation and "clips" values that would exceed a 16-bit representation. This routine was necessary because the existing arithmetic routines produce large negative values in the case of overflow. Overflow does not normally occur unless there is noise

ORIGINAL PAGE IS
OF POOR QUALITY

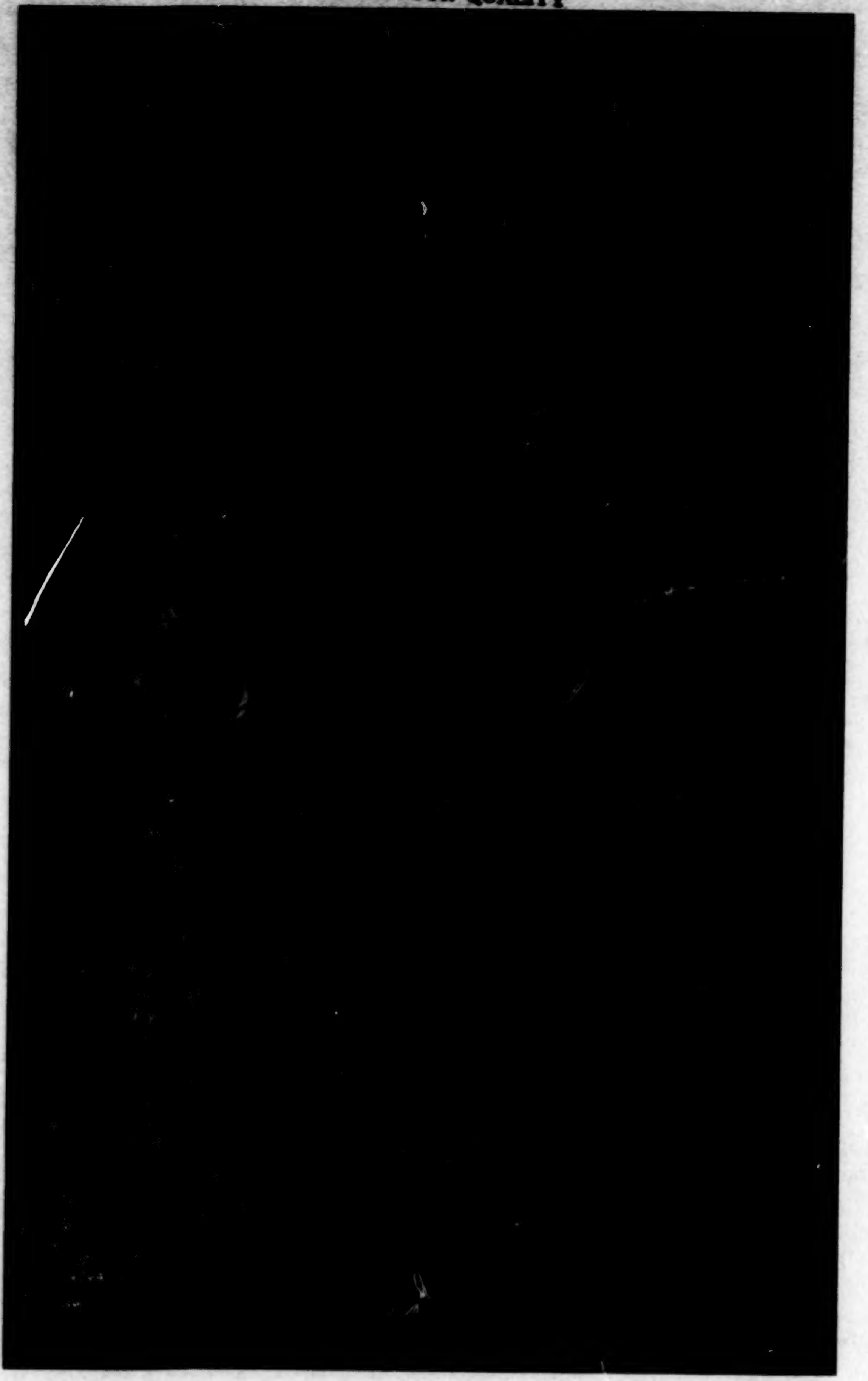


Figure 1. A SIR-B Image of Mount Shasta, California Generated Using the MPP

TIMINGS FOR PROCESSING 4096 x 4096 SIR-B SUBIMAGE

Processing	FFT Incorporating Scaling	FFT with no Scaling	Improved Code Generator
Range Compression	22.4	14.2	13.7
Azimuth Compression	9.3	5.9	5.3

I/O	Transposition Via 16 Bit Intermediate File	Transposition 8 Bit Intermediate File	Transposition in Stager	Estimate with Overhead Elimination
Raw Data Input	75.4	75.4	75.4	36.4
Constants and Params.	49.5	49.5	49.5	24.1
Transposition	225.0	124.1	62.0	1.4
Image Data Output	32.1	32.1	32.1	21.6

introduced while transferring data into the array unit. To avoid the effects of noise, the data values are rescaled at the intermediate stages of computation to provide maximum dynamic range, in effect increasing the signal-to-noise ratio. Given "well-behaved" data, these extra routines (which degrade the performance) are unnecessary. Each FFT requires 3.0 msecs, while without the clipping routine the time is reduced to 1.25 msecs. Further improvements in performance can be expected by optimizing the assembly code generated from the MPP Pascal source code.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the active participation, in the development of the signal processing algorithms on the MPP, by Drs. J. P. Strong and J. C. Tilton of GSFC, S. W. McCandless of the User Systems, Inc., J. A. Abeles of Science Applications Research, Inc., G. C. Floam and J. Hurst, formerly of Science Applications Research, Inc.

REFERENCES

1. Cumming, I. G. and Bennet, J. R., Digital Processing of Seasat SAR Data, MacDonald, Dettwiler and Associates, Ltd., 1979.

2. Fischer, J. R., Technical Summary of the Massively Parallel Processor, this volume.

3. Harris, F. J., On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings of the IEEE, 1978.

4. Raff, B. E. and Kerr, J. O., The Johns Hopkins University Applied Physics Laboratory SAR Processor, IEEE EASCON, 16th Annual Electronics and Aerospace Meeting, 1983.

5. Wu, C., A Digital Fast Correlation Approach to Produce Seasat SAR Imagery, Proceedings of the IEEE International Radar Conference, 1980.

6. Wu, C. et al, Modelling and a Correlation Algorithm for Spaceborne Signals, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-18, 1982.

N87-26546

PRECEDING PAGE BLANK NOT FILMED

AN APPLICATION OF THE MPP TO THE INTERACTIVE MANIPULATION OF STEREO IMAGES OF DIGITAL TERRAIN MODELS.

Sanjay Pol, David McAllister, Edward Davis
North Carolina State University
Raleigh, North Carolina
U.S.A

ABSTRACT

We develop MPP algorithms for the interactive manipulation of flat shaded digital terrain models defined over grids. The emphasis is on real time manipulation of stereo images. Standard graphics transformations are applied to a 128 x 128 grid of elevations followed by shading and a perspective projection to produce the right eye image. The surface is then rendered using a simple painter's algorithm for hidden surface removal. The left eye image is produced by rotating the surface 6 degrees about the viewer's y axis followed by a perspective projection and rendering of the image as described above. The left and right eye images are then presented on a graphics device using standard stereo technology. Performance evaluations and comparisons are presented.

Keywords: MPP, graphics, stereo images, parallel algorithms, terrain models.

INTRODUCTION

Grid based digital terrain models contain an m x n rectangular grid of points (x_i, y_j), $1 \le i \le m$, $1 \le j \le n$, which are longitude and laltitude values on the earth's surface. Each grid point has an associated value z_{ij} which is the elevation above sea level at the point (x_i, y_j). The data is often collected by cartographers from stereo photographs of the earth's surface using a stereoplotter. The cartographer uses input devices on the stereoplotter to manipulate a virtual 3D cursor. The operator drives the cursor across the surface of the earth while the stereo plotter produces digital elevations.

Although thousands of points may be collected on a given surface, these points are massaged to produce a grid of equally spaced terrain data. In a separate operation, data are collected describing features such as rivers, lakes,bridges, etc. The cartographers then search for anomalies or inconsistencies in the two sets of data. Anomalies might include rivers which run uphill, lakes which do not sit on the surface of the earth, etc.

The Defense Mapping Agency believes that anomalies in the data can be more quickly and easily located if cartographers are able to view and interact with these digital data using a three dimensional image. Interaction with the data in real time and the ability to shade the image require a high speed computer graphics device which can produce stereo images of the data at the refresh rate of the crt or projection mechanism. The technology used to produce and view such a three dimensional image is described in [1]. The left and right eye images share the odd and even scan lines of the crt, if interlaced refresh is used, while the two images are double buffered in graphics memory

PRECEDING PAGE BLANK NOT FILMED

and rendered alternately in the case of noninterlaced refresh. In both cases, a crt with a rapidly decaying phosphor is required to preclude ghosting. Polarization in the form of liquid crystal or plzt shutters is used to block the left eye when the right eye image is rendered and vice versa. The refresh rate must be sufficiently fast to eliminate flicker, usually at least 30 Hz interlaced or 60 Hz noninterlaced. In this paper the left and right eye images appear side by side. The stereo effect can be obtained by 'free viewing' the image. Free viewing requires the viewer to gently cross the eyes so that the two images merge to form the three dimensional image in the center. An alternative is to place a piece of paper perpendicular to the photograph so that the left eye sees only the left image and the right eye sees only the right image. Assuming a left handed coordinate system where x and y axes are centered on the viewing screen, the left eye view is formed by rotating the right eye view 6 degrees about the y axis. The two images are then drawn or rendered on the screen.

The points $P = \{(x_i, y_j, z_{ij})\}$ describe a discrete bivariate functional surface which can be rendered on a computer graphics device using standard polygon based graphics primitives. We discuss these primitives and their implementation in the next two sections. In later sections we present some examples and analyze the performace of the MPP and compare it to a VAX 11/780. Finally we offer directions for future research.

GRAPHICS PRIMITIVES

We assume m=n=128 and that the points in P are assigned to processors in the obvious way: processor p_{ij} contains the grid point (x_i, y_j, z_{ij}). Larger grids must be subdivided into 128x128 sections. The terrain surface is represented by triangles as shown in figure 1. Triangles have the advantage that they are always planar which simplifies rendering.

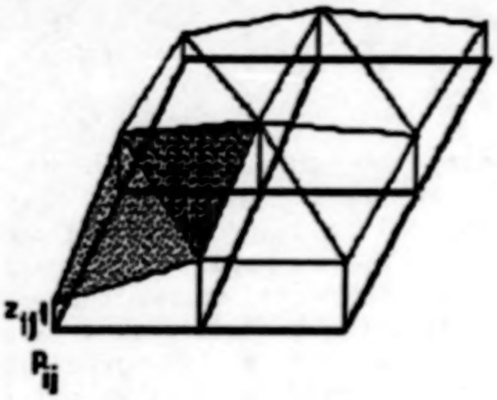


Figure 1. Terrain Model

The standard graphics operations of rotation, translation, scaling and perspective projection can be computed by matrix multiplications. Each point in P is multiplied by the transformation matrix to place the resulting terrain image in the proper position for viewing. The triangles comprising the terrain are then projected onto the viewing screen and filled with a shading intensity in an order which assures that triangles which are hidden from the viewer are eliminated. The latter process is called 'hidden surface elimination' [2].

Clipping involves presenting only that portion of the image that is visible to the user through his chosen viewing window, i.e. if the resulting image is too large to fit on the screen some of the image must be discarded before rendering. This process is computationally expensive on serial machines and we have chosen to ignore it here. We discuss it further in the final section.

The standard algorithms for hidden surface elimination are notoriously computation intensive on serial machines. They are designed to treat arbitrarily complex (and non-convex) objects which may intersect themselves, have holes, etc. However, the problem of hidden surface elimination can be solved easily for a digital terrain. The surface data can be divided into eight parts. The algorithm determines the octet number in which the chosen eye point lies based on the distances calculated to the four corners of the surface data. In any one octet the farthest point and the second farthest point are always fixed. (Figure 2).

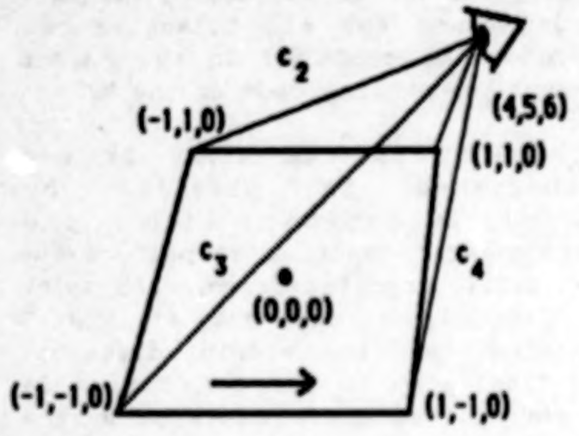


Figure 2. Rendering Order

The rendering of the terrain proceeds from the farthest to the second farthest corner. The technique is a painter's algorithm without a topological or geometric sort. For example, if the coordinates of the four corners of a rectangular grid centered at the origin are (1,1,0), (-1,1,0), (-1,-1,0), and (1,-1,0) the distances to these four corners from the eyepoint located at (4,5,6) are 25, 41, 61 and 45 respectively. The rendering therefore proceeds form corner 3 to corner 4. (Figure 2.). This technique is very effective in rendering digital terrains defined over rectangular grids. The approach is possible because a terrain can be rendered from back to front relative to the eyepoint and hidden surfaces will auomatically be removed. This algorithm makes possible real time manipulation of the stereo images because no sorting is required in the process.

With the emphasis on real time manipulation of stereo images, we have chosen to use the constant or flat shading model. In flat shading we assign the same intensity value to each pixel enclosed by a projected polygon. More sophisticated smooth shading techniques calculate separate intensities for each pixel within the polygon by interpolating the intensities across the edges of abutting polygons [3,4,5,6].

To compute the intensity or shading value to be assigned to a triangle two vectors are required: a vector of unit length perpendicular to the surface of the triangle called the unit normal and a unit vector from the centroid of the triangle to the light source (which we shall term the light vector) (Figure 3). The sun is the single light source.

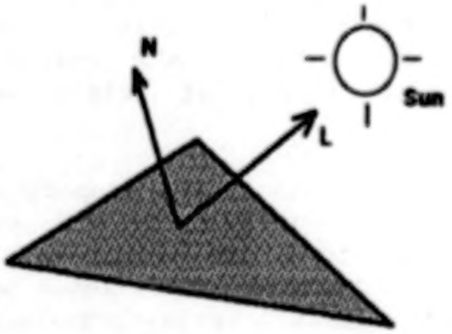


Figure 3. Flat Shading

The normal to the triangle is the cross product of any two sides of the triangle. The dot product of the normal and the light vector for the triangle is the cosine of the angle between the two and is used to determine the shade value used for illuminating the triangle. The distance to the light source is ignored. An intensity value representing ambient light is included so that all triangles are visible relative to the background color. We have not included a spectral reflection component since terrain images are not glossy.

Flat shading may result in a phenomenon called Mach banding which is the result of an anomaly in the human visual system which causes exaggeration of intensity change at any edge where there is a discontinuity in the slope of the intensity function [7]. This phenomenon is apparent in figures 4 and 5 in Color Plate I. It can be eliminated by resorting to smooth shading algorithms.

THE ALGORITHM

The algorithm which produces a stereo image is divided in two Sections. The first Section which executes on the MPP performs the necessary computations for correct orientation of the terrain surface. The second Section renders the terrain on the graphics device. All MPP related activities made use of references [8-14].

Section One contains as a subpart a preprocessing Fortran driver to produce input to a parallel Pascal routine. The Fortran driver reads or generates the digital terrain data and queries the user for the location of the eye point, and the location of the sun. The driver then constructs the necessary homogeneous transformation matrix which involves a series of matrix multiplications. The purpose of these transformations is to rotate the object so that the eyepoint lies on the Z axis. The scalar arrays for the eye point, light source, and rotation matrix are transferred to the MPP for computation of the stereo pair in Part 2 of Section One.

To compute the shade values to be assigned to each of the triangles, the normal to the surface of every triangle and the light vector are computed. These shade values are also output to the display part of the program. The total number of triangles generated for a 128 X 128 array is 32258 (127 x 127 x2). On the VAX the above computation must be done serially for each triangle. On the MPP, the computation for each triangle is assigned to one processing element. Shade values for all triangles can therefore be computed in two passes through the shading code on the MPP.

Each point in the array is now transformed in parallel by postmultiplcation with the transformation matrix. To perform the perspective transformation, each point is divided by the sum of its Z coordinate and the viewing distance. The final step in the computation is to convert the grid points to screen coordinates and to render the terrain on a graphics device.

EXAMPLES

Six stereo pairs are presented in figures 4 through 9 in the Color Plate I. All surfaces have been evaluated over a grid of 128 x 128 equally spaced points on [-1,-1] x [1,1]. In each case 256 intensity values have been used in the shading model.

Figure 4 on Color Plate I is a stereo pair of $f(x, y) = \exp[-(|x| + |y|) / 4] \cos(|x| + |y|)$. The image was chosen to demonstrate the hidden surface elimination procedure.

Mach banding is evident in this and the following figure. Figure 5 in Color Plate I is the function of figure 4 with the variable x removed. Figure 6 in Color Plate I is the usual cowboy hat function $f(x,y) = \exp(-(x^2 + y^2)) \cos(x^2 + y^2)$ while Figure 7 in Color Plate I is the function $f(x,y) = \exp(-|xy|) \cos(x^2 + y^2)$. Both surfaces appear to be smooth shaded.

Figures 8 and 9 in Color Plate I are digital terrain models representing a region in Seattle, Washington. The elevation values have been scaled up to exaggerate the depth in the image.

PERFORMANCE

The above terrain rendering algorithm has been implemented on the MPP at Goddard and on a VAX 11/780 at North Carolina State University. In one test case to compute the altitudes, perform the transformations and calculate the shade values for one of the two images the VAX used 10 seconds of CPU time. To display the data points on half of the screen on a Lexidata 90 graphics system in high resolution (1024 x 1280) requires 150 seconds of CPU time. This is the time to send the screen coordinates of the triangle vertices that form the image, their shade values and to render the image on the screen. We were not able to obtain performace statistics on the MPP because of a hardware error (failure to open Virtual Channel). However, performance estimates based on a 100 nanosecond time step for each processor, yielding a conservative overall execution rate of 200 Mflops for the PE's, shows that the computation of the transformations will require 0.02 seconds. Hence, computational aspects of real time stereo images indicate that it is possible to compute 50 stereo images per second for a 128 x 128 array on the MPP. The computation must be supported by appropriate high bandwidth I/O hardware and a fast rendering graphics device. The bandwidth required for rendering at a 60 frame per second rate using floating point data, is approximately 160 megabytes per second. Although the MPP-VAX system I/O rate is less than this, the MPP array interface is capable of this rate.

Computation rates and I/O rates can be reduced by using fixed point data with word lengths shorter than 32 bits. This is reasonable for the present problem because terrain data values can be represented in fixed point format. The highest peak on the surface of the earth is less than 32000 feet high. One foot resolution is usually sufficent which means that a 16 bit integer can be used for the elevation (z_{ij}) values. For a 128 X 128 array the longitude and latitude values can be represented as 8 bit integers. The MPP allows for these different data representations to exploit its computation, storage and I/O capabilities efficiently.

CONCLUSIONS AND FURTHER RESEARCH

Our initial studies show that grid based digital terrain models can be mapped in a natural way to the SIMD architecture of the MPP. The parallelism of the machine can be exploited in computing the necessary graphics transformations required to render stereo images of a terrain in real time on a high speed graphics device .

Development of the algorithm was inhibited by the memory limitations of the Processing Elements. The algorithm had to be divided into smaller segments to avoid overwriting on the memory bit planes used by the MPP. Additional memory will be required to handle larger grids.

Rapid identification of data anomalies may require closer inspection of parts of the terrain surface and hence zooming or scaling up of the image will be necessary. This may require clipping so the image fits on the viewing screen. Image enlargement may also require a pixel oriented smooth shading algorithm [3]. Shadowing can enhance visual realism and also aid in locating data anomalies. However, most shadowing algorithms on sequential machines are notoriously slow. Algorithms have been proposed which are adaptations of hidden surface techniques [15]. We intend to investigate these and other algorithms for possible implementation on the MPP.

REFERENCES

1. L.F.Hodges and D.F.McAllister, "Stereo and Alternating Pair Techniques for Display of Computer Generated Images", IEEE CG&A, Vol. 5, No.9, Sept.1985, pp 38-45.

2. I.E.Sutherland, R.F.Sproull and R.A.Schumacker,"A characterization of Ten Hidden-Surface Algorithms", Computing Surveys, Vol. 6, No.1, March 1974, pp. 1-55.

3. H. Gouraud, "Continuous Shading of Curved Surfaces",IEEE Transactions on Computers, Vol.C-20, No.6, June 1971, pp. 623-628.

4. Phong Bui-Tong,"Illumination for Computer Generated Pictures", Communications of the ACM, Vol.18, No. 6, June 1975, pp. 311-317.

5. K.E.Torrance and E.M.Sparrow, "Polarization, Direction Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces", J. Optical Society of America, vol. 56, No. 7, July 1966, pp. 916-925.

6. K.E. Torrance and E.M. Sparrow, "Theory for Off Specular Reflection from Roughened Surfaces", J. Optical Society of America, vol. 57, No. 9, Sept. 1967, pp.1105-1114.

7. F. Ratliff, Mach Bands: Quantitative Studies on Neural Networks in the Retina,Holden-Day, San Francisco, 1965.

8. General Description of the MPP, Goodyear Aerospace Corporation, GER 17140, April 1983.

9. MPP User's Guide, Edited by Jim Fischer, Science Applications Research,Lantham,MD., Jan. 1986.

10. Introduction to the MPP Host Run-Time System (MPPHRTS), Version 3.01, January 1986.

11. Parallel Pascal Language Reference Manual,MUD210, NASA, Goddard Space Flight Center.

12. Parallel Pascal User's Guide, MUD220, NASA, Goddard Space Flight Center.

13. MPP Pascal Callable Procedure Library, J. Abeles, Science Applications Research, Lantham, MD., January 1986.

14. Control and Debug (CAD) User's Manual, Goodyear Aerospace April 1983.

15. J.D.Foley & A.VanDam, Fundamentals of Interactive Computer Graphics, Addison Wesley, July 1984.

N87-26547

DEVELOPMENT OF A STEREO ANALYSIS ALGORITHM FOR GENERATING TOPOGRAPHIC MAPS USING INTERACTIVE TECHNIQUES ON THE MPP

James P. Strong
Space Data and Computing Division
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

The analysis of stereo images to determine depth or elevation is an example of the general problem of the detection of local movement or distortion when comparing two images. Examples of tasks that require solutions to this problem include topographical map generation, object tracking, autonomous vehicular guidance, and robotic vision systems. The key part of the problem is the matching of corresponding areas in the two images. While the human vision system is very good at this task, automated techniques have proven to be computationally expensive and not practical with standard computers. The thrust of the work presented in this paper is the development of a local area matching algorithm on the Massively Parallel Processor (MPP). It is an iterative technique that first matches coarse or low resolution areas and at each iteration performs matches of higher resolution. This is similar to what has been demonstrated to happen in human visual systems by Marr and Poggio, (1971). Results so far show that when good matches are possible in the two images, the MPP algorithm matches corresponding areas as well as a human observer. To aid in developing this algorithm, a control or shell program has been developed for the MPP that allows interactive experimentation with various parameters and procedures to be used in the matching process. (This would not be possible without the high speed of the MPP). With the system, optimal techniques can be developed for different types of matching problems.

INTRODUCTION

During October 1984, the Space Shuttle Challenger was flown with a Shuttle Imaging Radar instrument (SIR-B). One of the experiments during this mission was to obtain overlapping images of an area on the ground viewed from several different incidence angles. Any two of these images form "pseudo-stereo-pairs" which through a suitable geometric model can be used to compute surface elevations. The paper reports current results of an effort at the Goddard Space Flight Center to develop an automated algorithm for computing elevations from SIR-B image pairs using the Massively Parallel Processor.

Background

Historically, the derivation of elevations from stereo pairs has followed two general approaches: the contour and the profile. With the contour approach, the stereo pairs are adjusted in a viewer such that only objects at a certain height will overlap perfectly. The interpreter than traces the path of perfect overlap. In the profile approach, the spacing between the stereo pairs is adjusted until a given object overlaps perfectly in the two images. The height of the object is then obtained as a function of that spacing. Objects along a given line are usually matched with this technique and thus the term "profile". The second approach has been adopted for implementation on the MPP.

N87-26547

Difficulties in Stereo Matching

The major difficulties in detecting points or areas where perfect overlap occurs (i.e., matching of corresponding pixels in the two images) are:

1. Different brightness levels in the two images.
2. Local distortions of the image.
3. Low contrast areas and noise.

The first difficulty is often obviated by the use of normalized correlation functions for matching grey level or edge images. The second is inherent in stereo analysis because of the different viewing angles and makes automated matching more difficult than, for instance, in the case of matching control-point chips in Landsat images. It occurs most severely in regions of rapidly changing terrain and creates a horizontally stretched or compressed area surrounding corresponding pixels in one image relative to the other. Because of the large off-nadir viewing angle and difference in viewing angle required to achieve reasonable accuracy, this problem is particularly acute in radar images. Thus, the basic clue used to determine the elevation also makes the determination of that elevation more difficult.

Any techniques for correcting local distortions must take into account the fact that the distortion function can have a broad band of spatial frequencies. For example, the distortion function for a mountain range would have low frequencies, but added to these would be high frequencies caused by rock formations making up the surface. When a human observer fuses two images seen through a viewer, the low frequency information is used to obtain an initial fusion in which the eyes are brought into alignment (a technique used for automatic focusing of some cameras) and then high frequency information brings out a detailed perception of depth. The progression from low to high frequency suggests that a hierarchical approach for detecting matching pixels would be appropriate. With this approach, an initial match is performed on low frequency information in an image and then increasingly higher frequencies are incorporated to obtain the final matching of corresponding pixels.

Even with no local distortion, errors can occur due to noise, spatial periodicities and low contrast in the image. One way of reducing errors in general is to provide redundancy by computing matches at nearly every pixel in the image. Then continuity constraints on the ground surface can be used to correct local discontinuities in the elevation. Matching at nearly every pixel is a formidable task on standard serial computers. However, the architecture of the MPP is well suited to the local neighborhood operations required for matching pixels. The resulting speed allows iterations of the matching algorithm computed at every pixel in 512 x 512 images to be completed within seconds. The following sections discuss the matching technique developed for the MPP and results obtained using the MPP algorithm.

MATCHING TECHNIQUE

The matching algorithm developed for the MPP is an example of what has been termed the Hierarchical Warp Stereo (HWS) technique. Initial work on stereo analysis using a hierarchical approach was done by Marr and Poggio (1979). The Marr-Poggio algorithm performs low pass filtering and edge detection on the two stereo images and then matches the edges. Filters of several bandwidths are used from 1/100 to 1/4 the highest frequency. An edge in one image is said to match an edge in the other if 1) that edge appears within a given search area, 2) the slopes of the two edges match and 3) the direction of the change in brightness of the two filtered images is the same across the edge. The relative location of edges in the most highly (narrowest bandwidth) filtered image determines the relative positions of large objects in a scene (For instance, mountains or mountain ranges.) These displacements are then used to define search areas for corresponding edges in the second most highly filtered image. The procedure is repeated until corresponding edges are matched in the least filtered image.

More recent work has been reported by Quam (1984) who processes multiple resolution versions of both images (by sub-sampling by powers of 2 in both directions). Starting at the lowest resolution, matches are calculated using a normalized correlation measure applied to neighborhoods or windows surrounding reference and test pixels. The disparities between corresponding pixels (i.e., difference in their location in the two images) are then used as a one dimensional distortion function to warp one image (the test image) so that its matched pixels will be in the same location as in the other (the reference image). The warped image is then resampled at the next higher resolution. This cycle is repeated until the highest resolution images are matched. The warping operation at each iteration reduces the local distortion so that at the next iteration with the next higher resolution there is a higher probability of obtaining a good match between pixels. At the end of the process, the sum of the distortion functions from all iterations forms the disparity function used to compute elevations. Quam's algorithm also eliminates potentially bad matches at each iteration by interpolating across pixels with low values of maximum "match scores" obtained for the reference neighborhoods over the corresponding search areas in the test image.

MATCHING ALGORITHM ON THE MPP

The algorithm developed for the MPP is similar to the Quam algorithm with the following exceptions:

1. Instead of applying equal sized windows to versions of the input images at increasing resolution, decreasing sized windows are applied to each iteration to the original input images. This eliminates the need sub-sampling.

2. Iterations repeat until the neighborhoods are of a size within which there is no useful information for correlation.

3. At each iteration, the net amount of warping (i.e., an updated disparity function) is computed. This net disparity is always applied as the warping function to the original test image which eliminates loss of information at each warp iteration.

4. At each iteration, areas where bad matches occur are detected and interpolated over. Then the disparity function is smoothed before being used for warping.

The matching algorithm consists of the following steps:

1. Preprocessing of the test image,

2. Determination of matches,

3. Removal of "bad match" areas in the disparity function,

4. Smoothing the resulting disparity function,

5. Warping the test image.

The steps 2 through 5 are repeated for each iteration. The following subsections discuss each of these steps in detail.

Preprocessing of the Test Image

Because of the viewing geometry, the resolutions of the two SIR-B images are different in the stereopsis direction. Thus, a linear scale change is applied to the test image so that its resolution is the same as that of the reference image. This operation is implemented using a linear warping function in the stereopsis direction. Second, the test image can be translated to reduce the absolute value of the maximum disparity between the two images. Translation is effected by making the warping function constant over the image. When this is done, the size of the initial search area can be reduced. The amount of initial warping can be determined mathematically based on the different synthetic aperture radar incidence angles or it can be determined interactively by displaying the reference and test images. In either case, the initial warping function is incorporated into the net disparity function when determining elevations.

Determination of Matches

For each reference image pixel, a match is performed between a neighborhood surrounding that pixel (the "template") and neighborhoods within a search area in the test image. The location of the center pixel within each neighborhood in the test image relative to the reference pixel is the disparity value associated with that neighborhood.

The measure used for matching neighborhoods is the normalized mean and variance correlation given by:

$$\text{Match Score }(k) = \frac{\sum_i (X_i - \bar{X}) \times (Y_{i-k} - \bar{Y}_k)}{\text{SQRT}(\sum_i (X_i - \bar{X})^2) \times \text{SQRT}(\sum_i (Y_{i-k} - \bar{Y}_k)^2)}$$

Where X_i and Y_{i-k} are grey levels of the i^{th} pixels within the template neighborhood and the k-th in the search area respectively. The values X and Y_k are the mean values computed over the template and K-th search area neighborhoods.

For each pixel in the reference image, the match score for all neighborhoods within the search area is computed. The pixel at the center of the neighborhood with the highest correlation value or match score is selected as the matching pixel. The resulting disparity function is, therefore, made up of integer values.

Removal of "Bad Match" Areas in the Disparity Function

In order for stereo analysis to produce correct topographic results, there must be a one to one correspondence between pixels in the test image and those in the reference image (at least down to the resolution required to produce the desired elevation accuracy). For synthetic aperture radar images, this means that, 1) both images must be taken from the same side of the spacecraft (or aircraft) , and 2) both incidence angles must be such that there is no "layover" due to large surface slopes, and that there are no "shadows". If the image having the larger incidence angle is used as the reference image and both images meet the above two requirements, it can be proven that the disparity function will always have a gradient (or slope in the stereopsis direction) between 0 and 1. If "ground range" images are used (where both images have the same pixel resolution), the slope of the disparity function will always be between ± 1. This result is necessary if there is to be a one to one mapping between the test image and its warped version when the disparity is used as the warping function. Since the slope of the disparity function must be between ±1, a simple test for a bad match is to observe adjacent values of the disparity function and determine if there is a jump of more than 1 pixel.

The human visual system appears to have the capability of interpolating surfaces over areas where bad matches occur. This process is emulated in the MPP algorithm by interpolating the disparity function across all areas where a bad match has been detected. The detection of bad matches and the interpolation are accomplished with the following operations:

1. Detection of "bad match pixels" (pixels where discontinuities occur)

Sudden jumps in the disparity function are detected by examining a 3 x 3 neighborhood surrounding each pixel. If the disparity value at the pixel differs from that of any of its neighbors by more than one, the pixel is identified as having a bad match.

2. Expansion or growth of each bad match pixel to form a neighborhood

If the maximum difference between the disparity value at a pixel and those of its adjacent neighbors is "d", then one must interpolate the disparity function over a neighborhood surrounding that pixel whose diameter is at least "d" to satisfy the constraint that the slope of the disparity function be less than +1. The expansion of each "bad match pixel" to form a neighborhood is done for this purpose. This is accomplished on the MPP by alternately expanding each pixel into 4 and 8 element neighborhoods. The resulting neighborhood is octagonally shaped. A diameter of 2N is achieved with N iterations. As each pixel is expanded, the resulting overlapping neighborhoods form bad match regions.

3. Interpolation of the disparity function over resulting bad match neighborhoods

Interpolation is performed using heat flow equations. The architecture of the MPP is well suited to the iterative solution of the boundary value partial differential equations typical of heat flow problems. To perform the interpolation, two dimensional heat flow partial differential equations are applied to solve for the steady state "temperature" or disparity in the bad match regions assuming that the bordering pixels surrounding the bad match area are held at a constant "temperature" or disparity. The equations used for obtaining the interpolated disparity at pixel [i,j] at iteration t+1 from the values at iteration t are:

$$D(i,j,t+1) = D(i,j,t) + \frac{d(D(i,j,t))}{dt} \qquad \text{where}$$

$$\frac{d(D(i,j,t))}{dt} = \frac{\partial^2(D(i,j,t))}{\partial i^2} = \frac{\partial^2(D(i,j,t))}{\partial j^2}$$

The second partial differential equations reduce to

$$\frac{d(D(i,j,t))}{dt} = D(i,j+1,t) + D(i,j-1,t) + D(i+1,j,t) + D(i-1,i,t) - 4D(i,j,t)$$

when is is assumed that the two dimensional grid increments are unity. The number of iterations required to reach "steady state" is dependent on the size of the bad match regions.

Smoothing the Resulting Disparity Function

After interpolation, a smoothing operation is applied over the whole disparity function in order to obtain a smoother warping function with fractional pixel values rather than integer values. Smoothing is performed on the MPP by averaging over a neighborhood proportional in size of the neighborhood used to obtain the disparity function.

Warping the Test Image

The smoothed disparity function is used as a one dimensional distortion function to "geometrically correct" the test image in the stereopsis direction. The brightness values in the warped image are obtained by applying a linear interpolation function to the test image data in the resampling process.

INTERACTIVE OPERATIONS ON THE MPP

The matching algorithm on the MPP has been implemented to be run in an interactive mode where parameters such as neighborhood sizes, search area, and discontinuity thresholds can be input before starting each step of the matching algorithm. At the end of a given step, the results (such as the disparity function, or the warped version of the test image) can be immediately displayed or saved on disk. This interaction provides the ability to experiment with various parameter values and quickly observe the results. In addition, the control or shell program is designed so that operations can be easily modified or added.

The operations presently implemented in the matching algorithm which can be run interactively are shown below along with the input parameters which can be selected:

1. INITIAL WARP (left edge movement, right edge movement)
2. MATCH (neighborhood size, search area size)
3. "BAD MATCH DETECTOR" (discontinuity threshold for bad match, neighborhood diameter for expansion)
4. INTERPOLATE (number of iterations)
5. SMOOTH (neighborhood size)
6. WARP

The left edge and right edge movement in the INITIAL WARP operation define the linear warp function in the stereopsis direction. If they are equal, only a translation is applied to the test image.

Interactive Turn Around Time

The following table shows turn around times for the six stereo matching operations. Where times are dependent on parameters, some example parameters and the corresponding times are shown. The times are measured from the time a key is pressed on the terminal to start the task to the time the next prompt is displayed indicating that the task is finished. Times less than about a half second could not easily be measured.

The matching operation which is the most computationally expensive task has been optimized to require a time proportional to the length of the sides of the neighborhoods as opposed to the area. The smoothing operation is unoptimized and requires a time proportional to the area of the neighborhood. However, since the time required for smoothing is not prohibitively long for interactive purposes, this optimization has not yet been implemented.

INTERACTIVE TURN AROUND TIME FOR STEREO ANALYSIS OPERATIONS

Operation	Parameter		Time in Seconds
Initial Warp			0.5
Match	11 x 11	Nbh. size	5
	21 x 21		10
	41 x 41		20
Detect Bad Match	2	Radius	< 0.5
	4		0.5
	8		1
Interpolate	250	No. of Iterations	2
	500		4
	1000		8
Smooth	11 x 11		2
	21 x 21		6
	41 x 41		21
Warp			3

RESULTS AND CONCLUSIONS

The matching algorithm has been tested on overlapping SIR-B images with incidence angles of 25 and 42 degrees taken over a plateau region in Northern India at its border with Bangladesh. The signal to noise ratio is high almost everywhere in these two images. The results of the matching algorithm on these images are illustrated in Color Plates II & III. Figures 1a and 1b show the reference and test image. Viewing figures 1a and 1b stereoptically, one can observe the plateau region and the river valley around it. Figure 2a is the reference image again and figure 2b is the test image after it has been warped during the matching process. If one views figures 2a and 2b steroptically, it will be seen that there is virtually no depth since the warped test image matches the reference image very well. Two iterations of matching and warping were required to obtain this image. The first iteration used a 25 pixel square correlation neighborhood and the second, a 13 pixel square neighborhood. Further iterations with smaller neighborhoods 10 pixels square or less yielded too many discontuities. This indicates that for many areas in the particular SIR-B images used, there is insufficient information in neighborhoods smaller than about 10 pixels square to produce good correlation. Figure 3 is the two dimensional disparity function derived during the matching process. Dark areas in this image are where pixels in the reference image lie to the right of their correspondings pixel in the test image. In the light areas, the opposite is the case. The disparity function is approximately linearly proportional to the actual elevations in the images with the dark areas in figure 3 corresponding to low elevations and the lighter areas to higher elevations. In figure 4, a three-dimensional perspective view is presented of the disparity function. In generating this view, the disparity function was treated as a two dimensional surface illuminated by a light source at approximately the location of the shuttle radar sensor about 42 from vertical. One can easily identify the plateau region and river valleys corresponding to those seen in the original stereo pairs.

A close comparison of figures 2a and 2b shows that the matching portion of stereo image analysis for determining elevation can be accomplished with the MPP stereo matching algorithm as well as human observers in most areas where the local distortion is not too severe. A human capability which can possibly increase the resolution of the matching algorithm is the ability to discern and match edges practically to the nearest pixel. In areas where there are significant edges, one may be able to use smaller neighborhoods for correlation and thus, in these areas, increase the spatial resolution of the elevation data. The inclusion of edge detection and the matching of edges into the algorithm is one of the current changes being implemented on the MPP.

ACKNOWLEDGEMENTS

The author would like to acknowledge the help of Mr. William Haynes and Mr. Edward Seiler of Scientific Applications Research Corporation for aiding in the implementation of the matching algorithm on the MPP as well as Dr. H. K. Ramapriyan for his editorial contributions.

REFERENCES

1. Marr, D. and Poggio, T., "A Computational Theory of Human Stereo Vision", Proceedings, Royal Society of London, 1979, pp. 301-328.

2. Quam, L., "Heirarchical Warp Stereo", Proceedings, Image Understanding Workshop, 1984, pp. 149-155.

AFTERDINNER KEYNOTE SPEAKER

Dr. Ken Batcher
Goodyear Aerospace Corporation

N87-26548

The Architecture of Tomorrow's Massively Parallel Computer

Transcribed from an after-dinner talk given
by Dr. Ken Batcher
Goodyear Aerospace Corporation
on September 24, 1986

Goodyear Aerospace delivered the MPP to NASA/Goddard in May 1983, over three years ago. Ever since then we have tried to look in a forward direction. There is always some debate as to which way is forward when it comes to supercomputer architecture. In this talk, I will describe improvements to the MPP's massively parallel architecture in the areas of data I/O, memory capacity, connectivity, and indirect (or local) addressing.

I/O

Several years ago, Goodyear decided they should advertise the fact that they are something more than a tire company. They started a series of ads. A particular ad appeared in the *Wall Street Journal* a couple years ago saying that our computer can add and subtract 6 1/2 billion times a second (that's on eight bit additions). Someone at Goodyear thought up captions for the three men in the ad. The first man says "How long does it take to get the 6 1/2 billion pairs of numbers into the computer." The second man answers, "Oh, about a half an hour, then you add and subtract them in 1 second." So the third guy says, "Well, I hope they still make tires."

This points out the I/O problem of the MPP. Actually, the problem is shared by most supercomputers...they tend to be I/O bound. At the conference today, the speakers were saying that they were I/O bound. Figure 1 shows the rates at which data is transferred between various parts of the machine. The processing is going on between the PE registers and the ARU memory at 20,480 megabytes per second (assuming 16,384 wires pushing data at 100 nanoseconds per bit), so you can see the magnitude of the processing rate. Data slides in and out of the ARU memory from the side over 128, rather than 16,384 wires, so you get 160 megabytes per second, still a fairly respectable speed. In most computers if you turn on all the I/O, the processing would grind to a halt. It would take all the memory cycles. On the MPP you can run the I/O full bore and slow down the processing rate by 1.6%. The processing rate doesn't see the I/O.

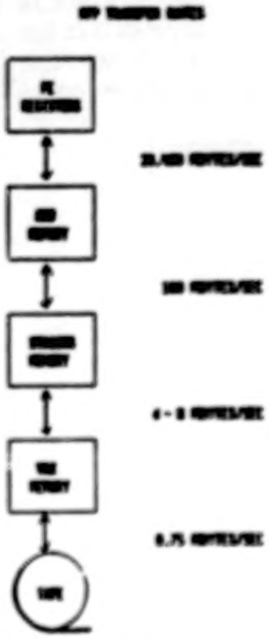


Figure 1

The I/O situation gets worse once we've put the data into the staging memory. Now we want to move it to the VAX computer through a DR780 channel, which is the fastest way you can get data in and out of a VAX but that's only at 6-8 megabytes per second. This is where the half hour figure comes from. You've got to put 6 1/2 billion pairs of numbers through this DR780 channel, which takes about half an hour, and the PEs would add them up in about 1 second. It's basically a limitation of the VAX. And it gets even worse when you look at what's the fastest way to get data in and out of the VAX, (unless you change the disk packs or something) you're limited to a tape drive that records at 6250 bits per inch---and 6250 bpi tape at 120 inches per second gives you .75 megabytes per

PRECEDING PAGE BLANK NOT FILMED

second. The significant difference between this .75 megabytes per second and the 160 megabytes per second rate out of the stager is basically your I/O problem.

You've got several ways of solving the problem, or at least making it less noticeable. We did design a disk farm for the MPP that would move data in and out of the staging memory at the staging memory rate of 160 megabytes per second. That's one possibility. The other possibility is some kind of high-speed network. If you had a device that was generating data at some fast rate, you could hook it directly into the staging memory at up to 160 megabytes per second. You want to bypass the VAX completely.

Another possibility is to increase the 160 megabyte per second rate to and from the array. Figure 2 shows the ARU as it exists right now. It has 128 columns, plus four spare columns. The data comes in from the left, goes out to the right, and is 128 wide at 160 megabytes per second input and output. If you aren't satisfied with that rate, you could divide the array up into four slices, put data into each slice simultaneously, and take it out of each slice simultaneously. This would give you four times the I/O rate, or 640 megabytes per second. If you want to preserve the redundancy feature of the array unit, you could add four spare columns to each slice of 32, so your array unit would have 144 columns in it, instead of 132.

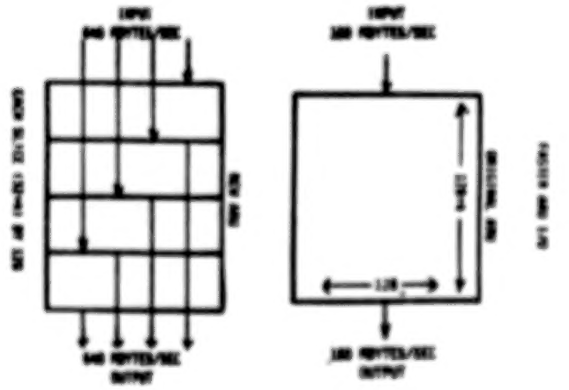


Figure 2

Memory Capacity

There seems to be a Parkinson's law of computer memory that says no matter how large you build the computer memory, there's always a computer problem that will overflow it. That's true of any computer, whether it's a personal microcomputer or a big supercomputer. The MPP is no different. Some of the speakers today talked about how they could use more memory...so there is a memory problem.

In the original specification back in 1979, NASA wanted 256 bits per PE for a total of half a megabyte of ARU memory (see table 3). We figured that was too small, so what we delivered was 1,024 bits, for a total of 2 megabytes of ARU memory. At that time we could get 4x1K static RAM chips with an access time of about 50 nanoseconds, and those are what we used to implement the MPP's ARU memory. We know that memory technology is always growing, so when we designed the machine we put in 16-bit addresses, so we could increase the ARU memory size later when the memory technology improved. Today, what we would do is build a board with memory sockets that could accept either 16K or 64K memory chips. Right now the 4x16K static RAM chips are readily available. Actually, the 4x64K RAMs are also available. They've got a high price tag but in a few years that should drop down. Today we could supply memory boards with either 16K or 64K bits per PE. This would increase the memory to either 32 or 128 megabytes. So you can get either 16 times or 64 times your present capacity—which is 2 megabytes. It's therefore real easy to expand ARU memory in the machine.

MPP ARU MEMORY

	BITS/PE	TOTAL (MBYTES)
ORIGINAL SPEC	256	0.5
DELIVERED IN 1983	1024	2
TODAY'S OPTIONS	16384	32
	65536	128

Table 3

In 1983, we delivered four banks of staging memory using 64K dynamic RAM chips, so the staging memory had a capacity of 2 megabytes and an I/O rate of 20 megabytes per second (see Table 4). We put 32 slots in the cabinet so that it could be expanded to 32 banks. This year we did expand it up to 16 banks. At the same time, we changed it to larger chips—256K chips. So right now the staging memory is 16 times larger than it was originally. The speed is 4 times greater. We still have half the slots available and at some time in the future we could expand it up to twice as big and twice as fast by populating all 32 slots. You can even go further than that; 1,024K (1 megabit) memory chips. These are starting to become available. So we could change the 32 boards and put in chips that are four times bigger and make the memory 256 megabytes. If you do that and you want the faster I/O in the ARU you've also got to feed it. You've got to take the staging memory and make it 128 banks...this will get the 640 megabytes per second. That will also give you a gigabyte of memory for the stager.

MPP STAGING MEMORY

	NUMBER OF BANKS	RAM CHIPS	CAPACITY (MEGABYTES)	I/O RATES (MBYTES/SEC)
1983	4	64K	2	20
1986	16	256K	33	80
FUTURE	32	256K	67	160
LARGER CHIPS	32	1024K	268	160
FASTER I/O	128	1024K	1073	640

Table 4

Connectivity

It is real easy to change the architecture of the machine for faster I/O and more memory and still be compatible with the current MPP. Thus, the modifications I have described so far are still upwardly compatible with the current MPP. Programs wouldn't have to be changed to use the larger memory capacity and faster I/O. On the other hand, modifications to connectivity and addressing could reduce upward compatibility. Someone defined upward compatibility as meaning that you get to keep all the old mistakes. So if you want to forget about being compatible and look at other changes in the machine, then we can talk about connectivity and indirect addressing.

Figure 5 is a picture of the current ARU. It has 16,384 processors and they communicate with each other in the north, south, east, and west directions over a 2-dimensional mesh. This is good for those problems in which communication must be nearest neighbor over a 2-D mesh. With the topology on the outside it can be changed into a 1-D mesh. And for 3-D problems you can always run a third dimension down the random access memory, especially if you make that memory larger...to say 65,536 bits. So, you can treat the 1-D, 2-D, and 3-D problems without too much trouble. However, there are a lot of problems that require other connectivities. We did add the staging memory, which in some respects helps because if you don't like the mesh connectivity, you can always move the data out to the staging memory, rearrange it and bring it back into the ARU so that items that used to be far apart are now close together. So this staging memory does help somewhat in giving you more connectivity than the 2-D mesh.

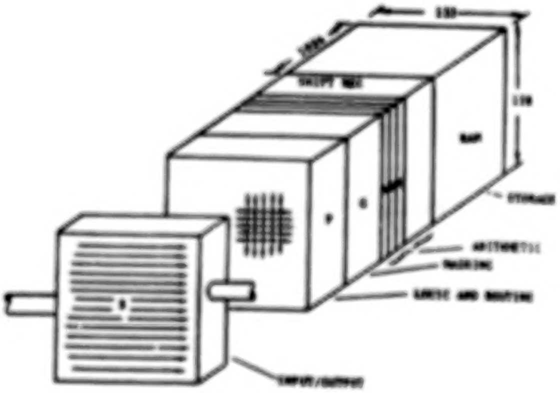


Figure 5

Looking back at the history of Goodyear Aerospace (see Table 6), some interesting trends in connectivity arise. Back in the mid 1960's we did a study contract for Griffis Air Force Base in Rome, New York, called *The Advanced Computer Organization Study*. We were looking at parallel processors that have 100 PEs in them. We figured that we wanted to hook anything to anything else and the only way we knew how to do this was with a sorting network—which in some respects is kind of an ideal—it will do any permutation by sorting.

[illegible]

		NO. OF PE'S	CONNECTIVITY
MID 60'S	ADVANCED COMPUTER ORGANIZATION STUDY	~100	SORTING NETWORK
EARLY 70'S	STARAN	512-1024	256-WIDE FLIP
EARLY 80'S	ASPRO	> 1024	32-WIDE FLIP
EARLY 80'S	MPP	16384	2D-MESH

Table 6

In the early 1970's we built some STARANs that consist of about 1,000 PEs and we connected them together with a 256 wide flip network, which is basically the same as an omega network or a butterfly network…it has the same topology. So the STARAN had less connectivity and more PEs than we were looking at in the mid '60's. Later, we built the ASPRO computer with about 1,800 PEs and a narrower flip network at 32 wide, that had less connectivity than the STARAN. We then delivered the MPP with 16,384 PEs and a 2-D mesh. When you look at this over time, we have been increasing the number of PEs and reducing the connectivity.

One way to explain the trend is to say "Well, back in the '60's we were young and more visionary than we are now. But now we are more practical and have more practical connectivities." I think the real trend is that each of these projects was in response to an RFP, and basically, we gave the customer what he wanted. Back in the mid '60's Rome was being more visionary, and now NASA is being more practical and asked for a 2-D mesh. This is an illustration of the Golden rule, "Whoever has the gold makes the rules." Whatever the customer wants is what we give them. That's what I think is the real explanation of this.

Over the years, we have been looking at connectivity networks other than the mesh. If you had your druthers, you would like to hook the 16,384 processors with a full cross bar (see Figure 7) so that any processor could talk to any number of its neighbors and could broadcast to any number of its neighbors. Unfortunately, that requires something like a quarter of a billion cross points, so it's rather impractical.

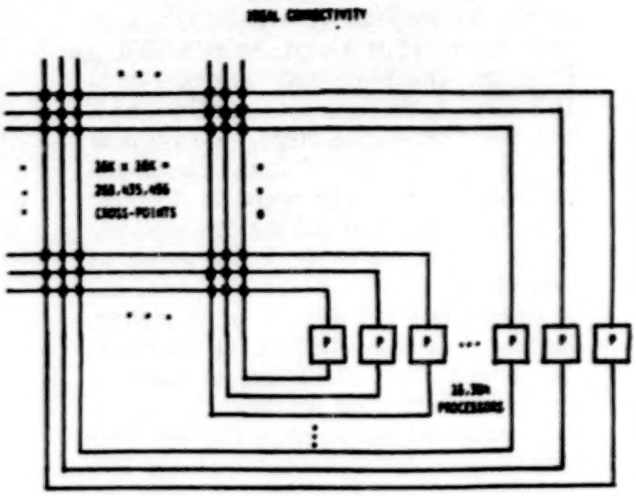


Figure 7

You can go various steps toward the ideal of full connectivity (see Table 8) using two approaches. You can take something like a flip network or an omega network…several people have talked about this network and they all give it different names. I call it a flip network, it has synonyms like omega network, butterfly network, delta network, etc.. They all have the same basic network topology, it's just different names. To connect *n* items together it takes $n/2(\log_2 n)$ switches. If you look at 16,384 processors, then it takes 114,688 switches, which is considerably less than the quarter billion needed for ideal connectivity. This does most of the useful permutations. For most problems this is what you want. If you want to be able to do any permutation of 16,384 items, then you would use a network that is about twice as big, requiring 221,184 switches. Unfortunately, it takes

INTERCONNECTION NETWORKS

- FLIP NETWORK

  $N/2 \ LOG_2N$ SWITCHES
  114,688 SWITCHES FOR 16,384 ITEMS

- BITONIC SORT NETWORK

  SELF-CONTROLLING
  $N/4 \ (LOG_2N)(1+LOG_2N)$ SWITCHES
  860,160 SWITCHES FOR 16,384 ITEMS

Table 8

awhile to compute how to set up such a network. If you want to go one step further you use the bitonic sort method, which in some respects is self controlling. It will compute the setting while the data is being fed into it. And it takes $n/4(\log_2 n)(1+\log_2 n)$ switches, which is 860,160 switches. So these are three ways of connecting the PEs, depending on what you want communicated between the processors.

The hypercube is basically somewhat like the flip network (or omega or butterfly), it has the same kind of complexity. These methods are the basic ways of improving the connectivity of the MPP, but then you lose something in compatibility. Currently your programs communicate north, east, west, and south, so you would have to do some work to use these other connection schemes.

Indirect (Local) Addressing

Figure 9 shows what one MPP PE looks like. The random access memory has a global address that comes from the control unit. In fact, if you look at how we implemented the PE, all the logic (except the memory) is on one chip, the PE chip, which doesn't even see the memory address. The only connections between the memory chip and the PE chip are the data paths. The global address goes to the memory chips directly and doesn't know that there is a PE chip at all. So the memory address is a global address. This means that if one PE wants to look at bit 43 in its memory, then all PEs must look at bit 43 in their memories. This means that when we program the machine, we look at it as a bunch of memory planes and processing planes. For example, you could take memory plane number 43 and move it from the memory into one of the processing planes or store a processing plane into a memory plane. All the data in one plane moves en masse in one cycle. So you look at the machine as a bunch of memory planes and processing planes. If you look at one bit of a plane, then you're looking at all bits of the plane.

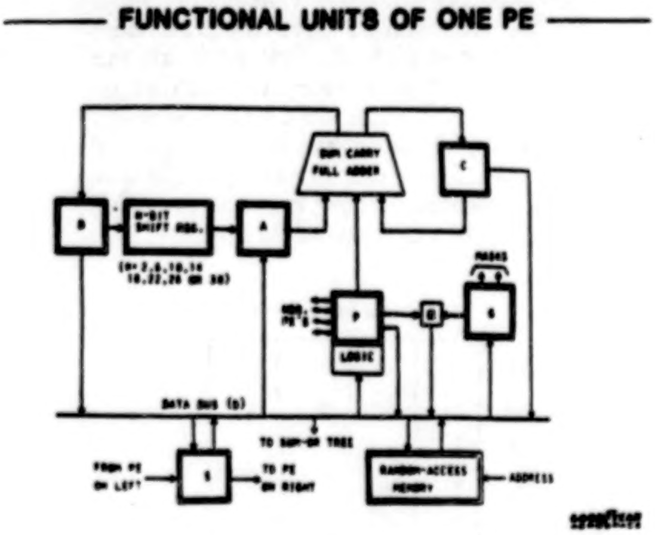


Figure 9

In all the textbooks the MPP and the Connection Machine are called SIMD machines---I question whether they are really SIMD machines. In the classic SIMD machine all the opcodes and addresses come from a common control unit (see Table 10) and this is true of the Connection Machine, the MPP, and any bit-serial parallel processor. Though all the textbooks call this SIMD, I can argue that it is really SISD, Single Instruction Single Data path. If you look at a conventional computer, you have a processor and memory and you take memory words from the memory into the processor and you store memory words. The only difference is that our memory words are 16,384 bits each. So you look at this MPP (see Figure 11), or the Connection Machine, as a conventional computer that happens to have a very large memory word, in this case 16,384 bits per memory word, and all we are really doing is moving memory words back and forth between processors and memory. So you can argue that the MPP and these other bit-serial parallel machines are really SISD machines. If you look at what a MIMD machine is; the opcodes and addresses come from local PE memories and registers. Each processor generates its own addresses and opcodes from programs stored in its own memory,

CLASSICAL
SIMD - OPCODES & ADDRESSES FROM COMMON
CONTROL UNIT (REALLY SISD).

MIMD - OPCODES & ADDRESSES FROM LOCAL PE
MEMORIES AND REGISTERS.

TRUE
SIMD - OPCODES FROM CONTROL UNIT.
ADDRESSES FROM LOCAL MEMORIES
AND REGISTERS.

Table 10

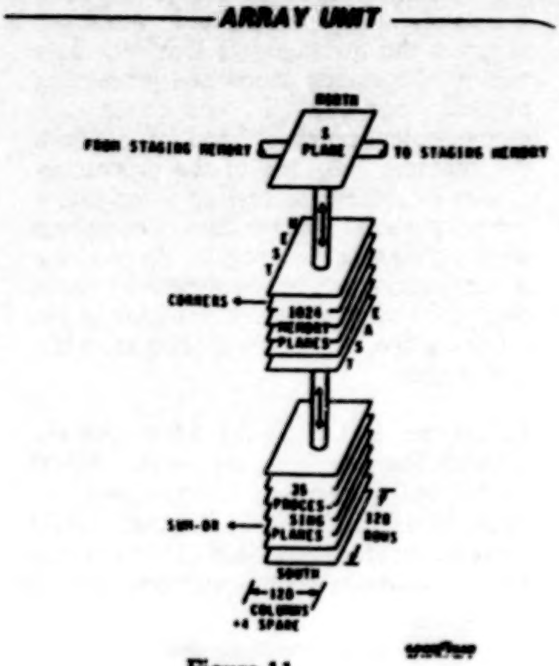


Figure 11

as opposed to the other case where they all come from a common control unit. So I can argue that a true SIMD machine would have the opcodes come from the control unit and still have a single instruction stream, however, if you really want multiple data, you should really have the addresses come from local memories and registers. This is somewhat in between the classic SIMD and MIMD machines. So, I think the true SIMD machine is one in which the addresses are generated locally, and the opcodes are generated in the control unit.

We found many problems where we really would like to address the memory via a local independent address rather than just through a global address. One such case comes up when trying to factor large numbers on the MPP. If we are trying to factor a particular large number, then what we do is look at factors of 1,000 to 4,000 quadratic residues. Table 12 shows an example of 4,000 quadratic residues. What we do is find the prime factors of each of these. We have maybe 4,000 primes in the horizontal dimension, and first we find what primes factor these residues, i.e., we try to divide them all by 2, then by 3, then by 5 and so on. We set up a flag matrix that shows where the division is exact. However, we are not done then. We want to take out, say, all the factors of 2, so if the number is 128, then we have to take out seven factors of 2 from that number. If it is 243, then we have to take out five factors of three. One way of doing this is to keep dividing by two until they are all odd, then divide by three as much as possible, and it can be seen that this is a sparse operation, i.e., most of the PEs are not participating. Each one of these divisions is in a separate PE, and most of them are not participating. We keep dividing out the higher powers of these primes. What we would like to do to improve the situation is to pack the data together so that we get the kind of arrangement shown in Table 13. We now start with let's say 15 columns for the prime factors. And then we divide these numbers by these numbers, do that until all the powers have been reduced and then we go on to the next column. We would rather do something 15 times than 4,000 times, and we would like to rearrange the data like that, pack it together where it is sparse. This improves the utilization of the machine.

4000 PRIMES →

← 4000 QUADRATIC RESIDUES

	2	3	5	7	11	13	17	19	23	29	31	
128	1	0	0	0	0	0	0	0	0	0	0	
600	1	1	1	0	0	0	0	0	1	0	0	
243	0	1	0	0	0	0	0	0	0	0	0	
[illegible]	0	0	0	0	0	0	0	0	0	1	1	. . .
[illegible]	0	0	0	0	0	1	1	0	0	0	0	
343	0	0	0	1	0	0	0	0	0	0	0	
[illegible]	0	0	0	0	1	0	0	1	0	0	0	
32	1	0	0	0	0	0	0	0	0	0	0	
⋮					⋮							⋱

Table 12

15 PRIME FACTORS →

← 4000 QUADRATIC RESIDUES

128	2				
600	2	3	5	25	
243	3				
[illegible]	29	31			. . .
[illegible]	13	17			
343	7				
[illegible]	11	19	43		
32	2				
⋮					

Table 13

When we were looking for a way to do this, we found that we could use the shift register (see Figure 9). The shift register was originally in there to improve the machine's times for multiplication, division, and floating point operations. However, it turns out that the shift register can be used for several other purposes. The shifting is maskable, so I can shift some shift registers and not shift others. So in effect, it turns out to be a locally addressable memory because I can turn the shifting of it on and off. Unfortunately, it only has 30 bits in it, I wish it were larger. So there is a small quantity of locally addressable memory in the MPP, and we have found out how to use it to help some of these problems.

To improve the MPP, I would probably replace the shift register with a RAM. You can simulate a shift register within a RAM and then you can do other things with the RAM. You would really like to make the whole memory locally addressable, but then you have some problems. If this is off on a separate chip, then you have to worry about how you transmit the address from the PE into the memory and back again. It either takes a lot of pins or some other circuitry. The compromise is to put a RAM on the chip and put another RAM off the chip. The on-chip RAM is implemented with VLSI techniques. Memory manufacturers make all their memory chips with their own rules to get a lot denser memory chips than what you can build just using the standard VLSI design rules. If you still want a lot of RAM, then you still need a standard memory chip from a manufacturer. Anyway, you could build your own locally addressable RAM. The places where you would like to see this is in Artificial Intelligence (AI).

I was looking at AI problems on the MPP. I found that this local addressability could be used to help out. For example, if several concepts are stored in each PE, while one PE is looking at its third concept, another PE may be looking at its fourth concept. With global addressing that is hard to do. With local addressability, it turns out to be a lot easier.

When I got Danny Hillis' book, the first thing I looked for was to see how they got local addressability...I couldn't find it anywhere. I know there are some Connection Machine people here, and I think that local addressability is a problem and I would like to see a newer machine have locally addressable RAM.

Conclusion

In conclusion, I talked about four topics (see Table 14). I/O---we can get transfer rates up to 640 megabytes per second. There are devices around that can supply the data and accept it at this rate. We can increase the memory capacity up to 128 megabytes in the ARU and over a gigabyte in the staging memory. For connectivity, there are several different kinds of multi-stage networks that we can consider. And we also should do something about local or indirect addressing.

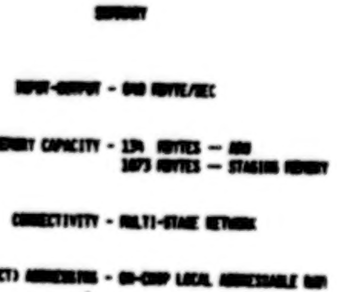


Table 14

EARTH SCIENCES SESSION II

Session Chair: Dr. Chester E. Grosch
ICASE, NASA Langley Research Center

PRECEDING PAGE BLANK NOT FILMED

N87-26549

IMPLEMENTATION OF THE BAROTROPIC VORTICITY EQUATION ON THE M P P

Max J. Suarez

NASA/GSFC/611
Greenbelt, Maryland 20771

and

Jim Abeles

Science Applications Research
4400 Forbes Blvd.
Lanham, Maryland 20706

ABSTRACT

A finite difference version of the equations governing two-dimensional, non-divergent flow on a sphere is implemented and integrated on the MPP. The MPP's performance is then compared with the CYBER's.

Keywords: Numerical weather prediction, computational fluid dynamics.

INTRODUCTION

The purpose of the work described here was to demonstrate the feasibility of using a massively parallel architecture to solve the hydrodynamic equations as they are used in numerical weather prediction (NWP).

Models used in NWP are commonly divided in two parts: the "dynamics" and the "physics". The dynamics performs the time integration of the equations of motion. The physics computes the heating, friction, and sources and sinks of water vapor. These two parts present very different problems to a highly parallel machine.

Many of the calculations in the dynamics involve the parallel updating of the many degrees of freedom allowed in the discretization and are thus very suitable to a machine like the MPP. Occasionally, however, it is necessary to obtain a spectral transform or solve an elliptic equation. These problems, although parallel, are non-local and thus difficult to implement efficiently on the MPP's nearest neighbor network. Fortunately, the non-local calculations can be minimized by a suitable choice of numerical scheme. For example, grid-point models, in which the equations are finite differenced in a latitude-longitude lattice, are much preferable to spectral models, which require frequent transformations between physical end spectral space. Still, non-

PRECEDING PAGE BLANK NOT FILMED

local calculations are not completely avoidable. In particular they appear in the solution of elliptic equations that occur when implicit time differencing schemes are used. Although these too could be avoided by using an explicit method (which is in fact done in many models, even on serial computers), we feel the architecture should not be so specialized as to completely forbid such choices.

Problems in the physics part of the codes are probably even more serious. In these, it is their non-parallel, rather than non-local, nature that makes for difficulties. As an example consider condensation. In most models this is done level by level, testing for super-saturation and passing the excess water to the next level below. That level in turn may become supersaturated, or may have been so already. The condensation calculation is then repeated and so on until "rainfall" reaches the surface. If parallelism is exploited by mapping each latitude-longitude point onto a different processor (this is really the only practical alternative in a machine with as many processors as the MPP), each one will in general encounter different condensation conditions. Processors at all grid points where there is no condensation, for example, will be idle in this segment of the code, and parallelism will be lost.

THIS STUDY

To start looking at the problems one faces with a parallel architecture, we decided to use the barotropic vorticity equation as a model of the "dynamics" part of NWP models. In this way we can test both the parallel grid-point updating segments and the more challenging problem of solving an elliptic equation.

At each step of the calculation we update the following equation for a new value of the vorticity:

$$\frac{\partial \zeta}{\partial t} = -\frac{u}{a\cos\varphi}\frac{\partial}{\partial\lambda}(\zeta+f) - \frac{v}{a}\frac{\partial}{\partial\varphi}(\zeta+f) \qquad (1)$$

where ζ is the vorticity, and u and v are the zonal and meridional velocity components of the non-divergent flow, φ and λ are the latitude and longitude, and f is the Coriolis parameter. As mentioned already, (1) is solved by finite-differencing on a latitude longitude grid. A leap-frog differencing scheme is used in time. Once a new value of the vorticity is obtained from the discrete version of (1), the Poisson equation:

$$\nabla^2\psi = \zeta \qquad (2)$$

is solved for the stream-function. To solve (2) we use a "fast" method in which the equations are first Fourier transformed in the zonal direction, then finite differenced in the meridional direction and solved as a set of tri-diagonal systems. The velocity components u and v are then obtained from

$$u = -\frac{1}{a}\frac{\partial\psi}{\partial\varphi}$$

$$v = \frac{1}{a\cos\varphi}\frac{\partial\psi}{\partial\lambda}$$

Having u and v, (1) can be updated again and the cycle completed.

To test the model, (1) was forced with sources of angular momentum and eddy vorticity, and damped by a linear drag.

Tests were conducted in parallel on the MPP and the CYBER 205 at Goddard Space Flight Center. The CYBER calculations were done with HALF-PRECISION (32-bit) arithmetic. Both MPP and CYBER codes were optimized for their machines to the best of our abilities; but both used exactly the same algorithm. In particular, the "fast" solver used for (2), which is very efficient on the CYBER, was retained on the MPP. On the other hand, a 128x128 square grid was used in both cases. This is optimal for the MPP. Higher resolution would require either doing a prohibitive amount of I/O, or keeping more than one grid-point per processor, which is not possible with the MPP's limited memory. The CYBER efficiency,in contrast, is independent of resolution for all practical choices.

RESULTS

The timing results are shown in Table I. We have separated these in two parts: the time spent solving the Poisson equation (2), and all the rest, which is mostly computing the right-hand-side of (1) and a little housekeeping. Units are msec./timestep. At the resolution used, we were taking 200 time steps per day. As may be seen, the code is approximately four times slower on the MPP than on the CYBER. This poor performance, however, is due entirely to the Poisson solver, which runs some ten times slower on the MPP. The updating of the vorticity equation is twice as fast on the MPP. This is a very encouraging result.

CODE	MPP	CYBER
UPDATING VORTICITY EQUATION	5.7	11.6
SOLVING POISSON EQUATION	65.0	6.3
TOTAL	70.7	17.9

TABLE I

If the NWP model is grid-point and uses explicit time differencing, the elliptic solver is not needed, and the MPP (or an MPP-like machine) should do very well in the dynamics. However, even if the model is implicit, and one or several elliptic equations have to be solved, the situation is not as bad as Table I would indicate. In a typical situation we would be solving some 40 equations like (1) (4 variables [u,v,T,q] at 10 levels), but at most 10 equations like (2). Using these figures, we can extrapolate our results to a full, grid-point, semi-implicit NWP model. This is shown in Table II. As may be seen, the situation is much improved; the MPP is now at near CYBER performance, even doing all ten vertical modes implicitly.

Obviously, much work remains to be done before massively parallel machines can be used efficiently for numerical weather prediction. In particular, it is imperative that much more parallel formulations and/or algorithms be developed for the physics codes, a problem we have not even begun to address here. Nevertheless, we feel that the results presented indicate a very real possibility of using MPP-like machines in NWP.

CODE	MPP	CYBER
UPDATING VORTICITY EQUATION	228	464
SOLVING POISSON EQUATION	650	63
TOTAL	878	527

TABLE II

N87-26550

CHEMICAL NETWORK PROBLEMS SOLVED ON NASA/GODDARD'S MASSIVELY PARALLEL PROCESSOR COMPUTER

Seog Y. Cho and Gregory R. Carmichael
Department of Chemical and Materials Engineering
University of Iowa
Iowa City, Iowa 52242

David M. Cohen and Mehmet H. Oguztuzun
Department of Computer Science
University of Iowa
Iowa City, Iowa 52242

ABSTRACT

The single instruction stream, multiple data stream MPP processor consists of 16,384 bit serial arithmetic processors configured as a 128 x 128 array whose speed can exceed that of current supercomputers (Cyber 205). This paper presents and discusses the applicability of the MPP for solving reaction network problems including the mapping of the calculation to the architecture, and CPU timing comparisons.

Keywords: Chemical Net Work Problems, Eulerian Transport/Chemistry Model, Air Pollution Model, Coupled ODE-IVP.

INTRODUCTION

A detailed model which describes the transport and removal of photochemical oxidants, and acidic species and precursors in the troposphere has been under development for the past nine years. The present analysis consists of about 30 coupled three-dimensional time-dependent non-linear partial differential equations and about 50-100 coupled non-linear ordinary differential equations.

The model is representative of a number of comprehensive Eulerian transport/chemistry models being developed for regional air pollution problems. However, these models are only feasible when run on "supercomputers". Our model was developed on the NASA-Langley CDC-STAR computer and is currently being exercised on the NASA-Langley Cyber 205, the NCAR CRAY-1 and a FAC M240 in Nagoya University, Japan. The execution times are 0.025

$$\frac{\text{CPU-sec}}{\text{grids-time step}}$$ on the FAC M240,

0.007 $$\frac{\text{CPU-sec}}{\text{grids-time step}}$$ on the

CRAY-1, and 0.70 $$\frac{\text{CPU-sec}}{\text{grids-time step}}$$ on a

VAX 11/780. Thus, a 24-hour simulation on the CRAY-1 for the eastern United States with 9500 grid points requires 100 CPU-minutes.

Our experience has shown that transport/chemistry models can execute about 70-100 times faster on the "supercomputers". However, 100 CPU-minutes/simulation-day is still too large for most applications. Since typical applications require simulations of seven to ten days. Therefore, to exercise these models various simplifying assumptions are used to decrease the CPU time. However, these assumptions add additional errors and uncertainties to the model results. Faster computers will enable the execution of the "best-science" model version.

Currently about 90% of the CPU time is

spent doing the chemistry calculations. The chemistry introduces the stiffness, the coupling, and the non-linearity into the model. Thus, the highest priority in continued model development is to search for ways in software and hardware to reduce the chemistry calculation. The purpose of this paper is to describe our attempts to exploit massively parallel computer architectures to accelerate the chemistry calculations.

MODEL OVERVIEW

The regional-scale combined transport/chemistry/deposition model is Eulerian and treats 50 chemical species. Thirty species are advected, while the remaining species are short-lived and are modeled using pseudo-steady state methods. The mathematical analysis consists of partial differential equations for the advected species and additional algebraic equations for the steady-state species. The advected species satisfy

$$\frac{\partial C_i}{\partial t} + \nabla (VC_i) = \nabla \cdot K \cdot \nabla C_i + R_i + S_i - G_i, \quad i = 1, \ldots, 30; \qquad (1)$$

where C_i is the gas-phase concentration of the ith chemical species, V is the wind velocity vector, K is the eddy diffusivity tensor, R_i denotes the chemical reaction term, S_i is the source term, and G_i is used to describe the mass transfer between the gas and condensed phases. The algebraic equations for the gas-phase species assumed to be at steady state are written as

$$R_i(C_1, C_2, \ldots, C_{50}) = 0, \quad i = 31, \ldots, 50. \qquad (2)$$

These equations are representations of general chemically reactive flow problems.

Simulation of regional transport, chemistry and deposition as described by Equations (1) and (2) requires numerical integration. The method presently used is a combination of the concept of fractional time steps and one-dimensional finite elements. This is referred to as Locally One-Dimensional, Finite-Element Method (LOD-FEM). The LOD procedures (Mitchell, 1969) split the multi-dimensional partial differential equation into time dependent, one-dimensional problems which are solved sequentially. The transport equations are solved using a Crank-Nicolson Galerkin finite element technique. Chemistry and mass transfer equations are solved using an adaptation of the semi-implicit Euler method proposed by Preussner and Brand (1981).

SCOPING STUDIES

Test Problem 1: Chemical Network Problem

To evaluate the ability of the MPP to calculate chemical network applications a simple four species test problem was selected. The four species (C_1, C_2, C_3, C_4) are involved in the following chemical reactions:

$$C_1 + C_2 \xrightarrow{k_1} C_3 \qquad (a)$$

$$C_3 \xrightarrow{k_1} C_1 + C_2 \qquad (b)$$

$$2C_2 \xrightarrow{k_3} C_4 \qquad (c)$$

$$C_4 \xrightarrow{k_4} 2C_2 \qquad (d)$$

The transport equations describing this system is represented by Eq(1) with i=1,2,3 and 4.

As mentioned in the model overview section, one way of numerically solving complex transport chemistry network problems is to split the equation into transport and chemistry parts. The chemistry calculations using this technique requires solving the set of equations

$$\frac{\partial C_i}{\partial t} = R_i \qquad i=1, \ldots, \# \text{ of species} \tag{5}$$

at each grid point in the discretized space.

The use of the semi-implicit Euler method to solve Eq(5) results in the equations

$$\frac{dC_i}{dt_r} = - C_i \sum_j d_i^j \prod_{k=1} C_k + \sum_\ell P_i^\ell \prod_m C_m \qquad i=1,\ldots 4 \tag{6}$$

This set of ODE-IVP's is solved within each transport time step, i.e.,

$t_o < t_r < t_{transport}$.

Now consider the case when we have 16,384 grid points in the discretized spatial domain. Therefore each chemical calculation within each transport step requires the solution of 16,384 sets of Eq(6). To implement the solution of these equations on the MPP requires first the choice of how to map the equations to the architecture. In this case we have chosen to simply view each processor as a grid point in the discretized space, and to have each processor solve its own set of Eq(6). The algorithm for solution of Eq(6) is written in Parallel Pascal and resides on the VAX. The initial conditions and constants are distributed to each processor and the algorithm is executed on the MPP and output is sent back to the VAX.

The CPU time required for execution of 100 time steps on the MPP of this 4 species mechanism at 16,384 grid points is 0.293 CPU-seconds. The same problem was executed on the VAX-11/780 and required 138 CPU-seconds. Thus for this chemical network problem the MPP executed a factor of 470 times faster than VAX 11/780!!

Test Problem 2: NO_x Transport in Eastern United States

To test combined transport/chemistry network problems on the MPP a 3-dimensional test problem describing the transport and chemistry of NO, NO_2, O_3, and HNO_3 in the lower troposphere was selected. The governing set of equations is given by Eqn. (1). An oversimplified chemical mechanism is used in this test calculation, i.e.,

$$NO + O_3 \rightarrow NO_2 + O_2 \tag{e}$$

$$NO_2 + h\nu \xrightarrow[+O_2]{} NO + O_3 \tag{f}$$

$$NO_2 + OH \rightarrow HNO_3 \tag{g}$$

(The OH concentrations are given by an empirical formula and the O_2 concentrations are assumed constant.

Sample results for this test problem calculated on a VAX 11/780 are shown in Figures 1 and 2. Presented are the NO_x emissions for the eastern United States, and the 24-hour averaged predicted surface concentrations of NO, NO_2, and HNO_3. The meteorological conditions simulated are those of July 4, 1974. The grid system used in the simulation was 32 x 32 x 16, and a transport time step of 15 minutes and a chemistry time step of 1 second was used.

This combined transport/chemistry problem is currently being run on the MPP. There are two choices for mapping this problem to the MPP. One method is to perform the chemistry calculations on the MPP and the transport part of the calculation on the VAX. This

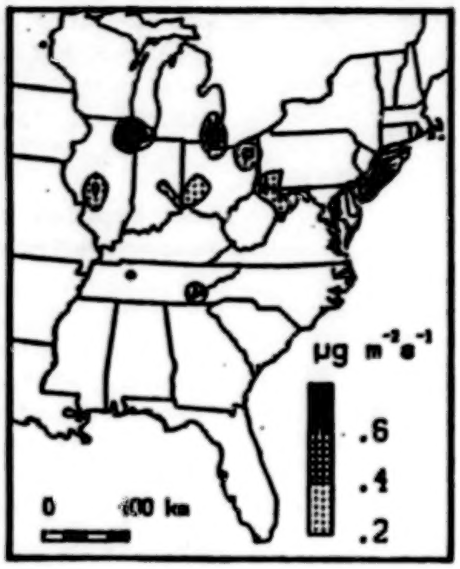
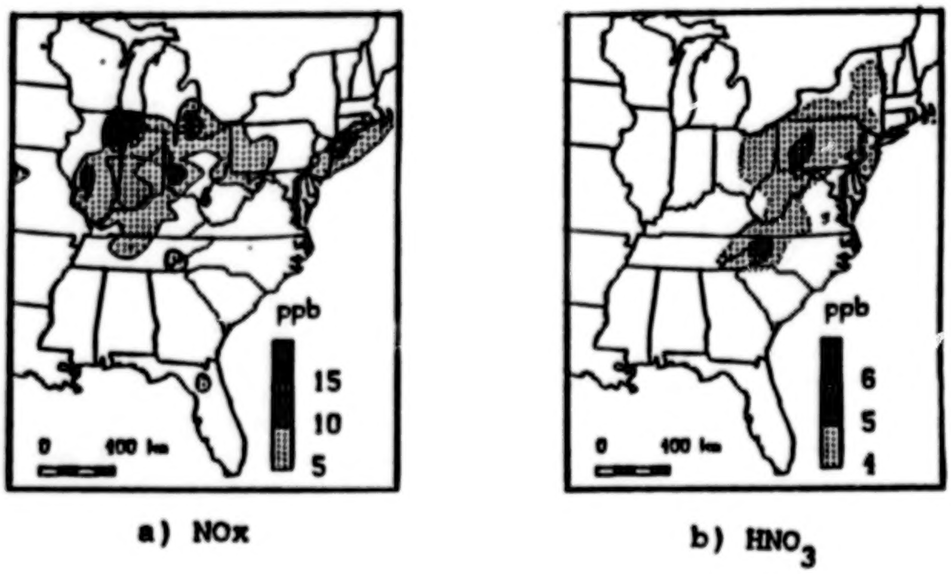

μg m⁻² s⁻¹
.6
.4
.2
0
400 km

Figure 1. The emission of NO_x at July 4, 1974 at surface.


ppb
15
10
5
0
400 km
ppb
6
5
4
0
400 km

a) NOx

b) HNO_3

Figure 2. Averaged concentration of July 4, 1974 at surface.

method is currently being tested. The other method is to perform the entire calculation on the MPP. We are currently developing an algorithm to solve the sets of tridiagonal matrices on the MPP which arise from the transport processes.

The above test calculation indicates the MPP is well suited for chemical network problems where each node can hold the entire mechanism. Current memory restrictions limit the size of the chemical mechanism that can be solved in this fashion. At present each processor can hold 32 32-bit variables. (It is planned to increase the storage in the near future.) However, it is possible to handle larger chemical mechanisms. One way is to group processors together. For example if 128 words are required at each node then four processors can work together. This in turn would reduce the maximum number of grids possible by a factor of 4. Another way is to make use of the staging memory.

SUMMARY

The suitablity of the MPP computer for calculation of chemical network problems is under evaluation. To date the MPP has been used to calculate a test problem which represents one component of a sophisticated chemically reactive flow problem. Specifically the set of coupled ODE-IVP's describing the chemical reactions occuring at 16,384 spatial grid points was calculated. This problem is ideally suited for the MPP because by using operator splitting, the chemistry at each grid point acts independently from that of the other grids (within each transport time step). This test calculation showed that the MPP can perform a 100-time-step calculation 470 times faster than the same calculation on the VAX 11/780. Also since nearly 90% of CPU time of large chemically-reactive flow problems is spent doing the chemistry calculations, the MPP architecture offers great potential for CPU savings for model applications. Coupled transport-chemistry problems are now being tested on the MPP.

ACKNOWLEDGMENTS

The authors wish to thank the staff of NASA/Goddard, and specifically Jim Abeles, for their assistance in helping us use the MPP, the University of Iowa Graduate College for providing funds to carry out this project, and Chi-Chul Yang, a graduate assistant in the Department of Computer Science for assisting with MPP programming.

REFERENCES

1. Gabriel, R. P. "Massively Parallel Computers: The Connection Machine and NON-VON", Science, 231, 975-978, 1986.

2. Hwang, K. and Briggs, F. A. Computer Architecture and Parallel Processing, McGraw-Hill, New York, Table 6.3, 425, 1984.

3. Lerner, E. J. "Parallel Processing Gets Down to Business", High Technology, July, 1985, pages 20-28.

4. Mitchell A. R. Computational Methods in Partial Differential Equations, John Wiley, New York, 1969.

5. Preussner P. R. and Brand K. P. "Application of a Semi-Implicit Euler Method to Mass Action Kinetics", Chem Eng. Sci., 10, 1633-1641, 1981.

N87-26551

CONTEXTUAL CLASSIFICATION ON THE MASSIVELY PARALLEL PROCESSOR

James C. Tilton
NASA Goddard Space Flight Center
Greenbelt,MD 20771, U.S.A.

ABSTRACT

Classifiers are often used to produce land cover maps from multispectral earth observation imagery. Conventionally, these classifiers have been designed to exploit the spectral (and, for multi-date data sets, temporal) information contained in the imagery. Very few classifiers exploit the spatial information content of the imagery, and the few that do rarely exploit spatial information content in conjunction with spectral and/or temporal information. We are studying a contextual classifier that exploits spatial and spectral information in combination through a general statistical approach. Early test results obtained from an implementation of the classifier on a VAX-11/780 minicomputer were encouraging, but they are of limited meaning because they were produced from small (50-by-50 pixel) data sets. Here we present an implementation of the contextual classifier on the Massively Parallel Processor (MPP) at the Goddard Space Flight Center (GSFC) that for the first time makes feasible the testing of the classifier on large data sets.

Keywords: Image classification, image pattern recognition, image contextual analysis, parallel processing, earth remote sensing.

INTRODUCTION

Algorithms that are currently used in most multispectral classification studies are unable to exploit the full spatial resolution of the Thematic Mapper (TM) data. Paradoxically, these algorithms often produce more accurate classifications if the spatial resolution is degraded from 30 meters to the 80 meter resolution of Multispectral Scanner (MSS) data (Refs. 1,2), whereas humans can visually identify features more accurately in TM data at its original spatial resolution. This paradox is explained by noting that humans routinely use spatial information to help identify features in an image, while current commonly used classification algorithms do not use spatial information at all. The contextual classifier discussed here, however, does exploit spatial information, and has the potential of producing more accurate classifications of TM imagery at full resolution.

This contextual classifier was developed at Purdue University (Refs. 3,4), but it was tested only on 50-by-50 pixel data sets. The results produced in these tests were encouraging, but they were of limited value because of the small size of the test data sets. The classifier was not tested on larger data sets because it took too long to run on a VAX-11/780 minicomputer.

Testing the contextual classifier on large data sets becomes feasible when the algorithm is implemented on a massively (or fine-grained) parallel computer. Such a parallel computer is the Massively Parallel Processor (MPP) at the NASA Goddard Space Flight Center. The MPP is a Single Instruction, Multiple Data stream (SIMD) computer which was built by Goodyear Aerospace for the NASA Goddard Space Flight Center (Refs. 5,6). It consists of 16,384 bit serial microprocessors connected in a

PRECEDING PAGE BLANK NOT FILMED

128-by-128 mesh array with each element have data transfer connections with its four nearest neighbors. With this architecture, the MPP is capable of billions of operations per second.

A version of the contextual classifier has been implemented on the MPP, and a test of the classifier on the MPP took a total of 5 minutes to produce a 120-by-120 pixel classification. It would take roughly 12 hours to perform the same classification on a VAX-11/780 minicomputer. A 512-by-512 pixel classification takes one to two hours on the MPP (depending on parameter settings), whereas it would take one to two weeks to complete on a VAX-11/780 minicomputer. This more than a 100-fold improvement in running time has been obtained with a program written in a high level language on the MPP (MPP Pascal) with no concerted effort to optimize the program. We anticipate an additional 5 to 10-fold improvement in program running time with a highly optimized version of the program on the MPP.

We first present a derivation of the contextual classification decision rule, followed by a description of the implementation of the contextual classifier on the MPP. We close with some preliminary test results.

DERIVATION OF THE CONTEXTUAL CLASSIFICATION DECISION RULE

In the contextual approach to classification, the probable classifications of neighboring pixels influence the classification of each pixel. Classification accuracies can be improved through this approach since certain ground-cover classes naturally tend to occur more frequently in some contexts than in others. The contextual classifier that we have implemented on the MPP is the algorithm formulated by Swain et al (Ref. 3) and further developed by Tilton et al (Ref. 4). Here compound decision theory is invoked to develop a classification method which exploits spectral and spatial information.

The derivation of the decision rule for the contextual classifier assumes that the data can be modeled as a two-dimensional array of $N = N_1 \times N_2$ of picture elements (pixels). At each pixel location (i,j) we are given an n-dimensional observation X_{ij} which is assumed to be a random sample from a distribution characteristic of the fixed but unknown true classification θ_{ij}. The observation X_{ij} usually contains spectral and/or temporal information about the pixel location (i,j), and the classification θ_{ij} can be any one of m spectral or ground cover classes from the set $\Omega = \{\omega_i\}$, $i = 1, 2, \ldots, m$.

In its most general form, the theory allows for a decision rule that is different for each pixel in the image, and, for each pixel, depends on the context of the entire image, $\mathbf{X} = \{X_{ij} | i=1,2,\ldots,N_1; j=1,2,\ldots,N_2\}$. To obtain a tractable decision rule, however, we restrict the decision rule to be fixed for the entire image, and the context to be a subset of the entire image.

Define the context of the pixel at location (i,j) as p-1 observations spatially near, but not necessarily adjacent to, the observation X_{ij}. These p-1 contextual observations are taken from the same spatial positions relative to pixel position (i,j) for all i and j. Call this arrangement of pixels together with X_{ij} the p-context array. (A common p-context array for p=5 would be the observation X_{ij} at pixel (i,j) and the observations at the four nearest neighbor locations to pixel (i,j).) Group the p observations in the p-context array into a vector of observations $\mathbf{X}_{ij} = (X_1, X_2, \ldots, X_p)^T$ and let $\boldsymbol{\theta}_{ij}$ be the vector of true but unknown classifications associated with the observation $\mathbf{X}_{ij}$. Let $\theta^p \varepsilon \Omega^p$ and $\mathbf{X}^p \varepsilon (R^n)^p$ stand respectively for p-dimensional vectors of classes and n-dimensional measurements; each component of θ^p is a variable which can take on any classification value $\Omega = \{\omega_i\}$, $i = 1, 2, \ldots, m$; each component of $\mathbf{X}^p$ is a random n-dimensional vector which can take on values in the observation space. Correspondence of the components of $\mathbf{X}_{ij}$, θ_{ij}, $\mathbf{X}^p$, and θ^p to the positions in the p-context array is fixed but arbitrary, except that the pth component always corresponds to the pixel being classified.

We can now develop a decision rule, $d(\mathbf{X}_{ij})$, which assigns a minimum risk classification to pixel (i,j) based on the vector of observations $\mathbf{X}_{ij}$. The loss suffered by making the classification decision $d(\mathbf{X}_{ij})$ for pixel (i,j) when the true class is θ_{ij} is denoted by $\lambda(\theta_{ij}, d(\mathbf{X}_{ij}))$ for some fixed non-negative function $\lambda(\cdot,\cdot)$. The expected average loss (or risk) over the entire image is then

$$
\begin{aligned}
R_\theta &= E\left[\frac{1}{N}\sum_{i,j}\lambda(\theta_{ij}, d(\mathbf{X}_{ij}))\right] \\
&= \frac{1}{N}\sum_{i,j} E[\lambda(\theta_{ij}, d(\mathbf{X}_{ij}))] \\
&= \frac{1}{N}\sum_{\theta^p \varepsilon \Omega^p}\ \sum_{\substack{i,j \text{ with} \\ \theta_{ij}=\theta_p}} E[\lambda(\theta_p, d(\mathbf{X}_{ij}))] \\
&= \sum_{\theta^p \varepsilon \Omega^p} \frac{1}{N} \sum_{\substack{i,j \text{ with} \\ \theta_{ij}=\theta_p}} \int \lambda(\theta_p, d(\mathbf{X}^p)) f(\mathbf{X}^p|\theta^p)\, d\mathbf{X}^p \\
&= \sum_{\theta^p \varepsilon \Omega^p} G(\theta^p) \int \lambda(\theta_p, d(\mathbf{X}^p)) f(\mathbf{X}^p|\theta^p)\, d\mathbf{X}^p \\
&= \int \sum_{\theta^p \varepsilon \Omega^p} G(\theta^p)\lambda(\theta_p, d(\mathbf{X}^p)) f(\mathbf{X}^p|\theta^p)\, d\mathbf{X}^p \qquad (1)
\end{aligned}
$$

where θ_p is the p^{th} component of θ^p, and $G(\theta^p)$, the **context function**, is the relative frequency with which θ^p occurs in the array θ. For any array θ, a decision rule $d(\mathbf{X}^p)$ minimizing R_θ can be obtained by minimizing the integrand of Equation 1 for each $\mathbf{X}^p$; thus for a specific $\mathbf{X}_{ij}$ (an instance of $\mathbf{X}^p$), an optimal action is:

$d(\mathbf{X}_{ij})$ = the action (classification) α which minimizes

$$\sum_{\theta^p \varepsilon \Omega^p} G(\theta^p)\lambda(\theta_p, \alpha) f(\mathbf{X}_{ij}|\theta^p). \qquad (2)$$

In practice, a "0-1 loss function" is employed, giving

$$\lambda(\theta, \alpha) = \begin{cases} 0, & \text{if } \theta = \alpha \\ 1, & \text{if } \theta \neq \alpha. \end{cases}$$

Then Equation 2 simplifies, and the decision rule becomes:

$d(\mathbf{X}_{ij})$ = the action (classification) α which **maximizes**

$$\sum_{\substack{\theta^p \varepsilon \Omega^p, \\ \theta_p=\alpha}} G(\theta^p) f(\mathbf{X}_{ij}|\theta^p). \qquad (3)$$

A further assumption we make at this point is class-conditional independence of the observations (pixels) for any observation vector $\mathbf{X}_{ij}$. In this case,

$$f(\mathbf{X}_{ij}|\theta^p) = \prod_{k=1}^{p} f(X_k|\theta_k) \qquad (4)$$

where X_k and θ_k are the k^{th} elements of X_{ij} and θ^P, respectively. Evidence that this is a reasonable assumption for Landsat MSS data may be found in Ref. 7. Invoking the class-conditional independence assumption, the decision rule (Equation 3) becomes:

$d(X_{ij})$ = the action (classification) α which maximizes

$$\sum_{\substack{\theta^P \in \Theta^P, \\ \theta_p = \alpha}} G(\theta^P) \prod_{k=1}^{p} f(X_k|\theta_k). \qquad (5)$$

Methods for estimating the context function $G(\theta^P)$ are discussed in Ref. 4. We use the "unbiased estimator", which is the most flexible and successful of these methods. Using this method, we first generate an unbiased estimate of a priori probabilities for each class at each position in the context array using the method described in Ref. 4. The product of these a priori probabilities is then calculated over the context array, forming the unbiased estimate of $G(\theta^P)$ based on one image point. The final estimate of $G(\theta^P)$ is made by averaging the individual point estimates over a portion of the data.

Conventional multispectral classifiers often classify into spectral classes (spectrally differentiable subclasses) rather than directly into the ground cover classes of interest. The spectral class classification is normally renumbered in a post-processing step to produce a classification map in terms of the ground cover classes. When the classification is done in terms of spectral classes, we assume that $f(X_k|\theta_k)$ is a multivariate normal density with mean vector and covariance matrix determined by the class, θ_k. In the case where the classification is done in terms of ground cover classes, we assume that $f(X_k|\theta_k)$ is a weighted sum of multivariate normal densities, viz.

$$f(X_k|\theta_k) = \sum_{\zeta_k \in \theta_k} r(\zeta_k|\theta_k) g(X_k|\zeta_k) \qquad (6)$$

where ζ_k is the k^{th} spectral class, $r(\zeta_k|\theta_k)$ is the conditional probability of spectral class ζ_k given ground cover class θ_k and $g(X_k|\zeta_k)$ is a multivariate normal density with mean vector and covariance matrix determined by the spectral class, ζ_k.

IMPLEMENTATION OF THE CONTEXTUAL CLASSIFIER ON THE MPP

In both the parallel MPP implementation, and the conventional serial implementation, classification directly into ground cover classes generally requires significantly less computer time than a classification into spectral classes (Ref. 4). Let m be the number of ground cover classes, c be the number of spectral classes ($c \geq m$), and p be the number of pixels in the p-context array. If, for example, c=2m, a contextual classification into spectral classes would have to consider $(2m)^p$ context configurations, while a contextual classification directly into ground cover classes would only have to consider m^p context configurations. If the classification is performed using four nearest neighbor context (i.e., p=5), then the spectral class classification would pass through the main loop in the contextual classification program a (multiplicative) factor of 32 times the number of passes that would be required for a ground cover class classification. Since the ratio of spectral classes to ground cover classes is often greater than 1.5 or so, we normally classify directly into ground cover classes with the contextual classifier.

Since the training classes are nearly always given as a set of multivariate normal distributions corresponding to spectral classes (in this case, the $g(X_k|\zeta_k)$ in Equation 6), we must first estimate the $r(\zeta_k|\theta_k)$ in Equation 6 in order to calculate the $f(X_k|\theta_k)$ used in the contextual classification decision rule, Equation 5. In our implementation, the same unbiased estimator used to estimate the a priori probabilities for the context function is used to estimate the $r(\zeta_k|\theta_k)$ by limiting the classes ζ_k to the spectral classes associated with ground cover class θ_k. This step can be considered to be a preprocessing step, and is in fact implemented as a separate MPP program. In our implementation, we use the MPP to calculate the average value of $g(X_k|\zeta_k)$ for each ζ_k over the entire data set (the program cycles through as many 128-by-128 pixel sections of data as required to cover the entire data set), and return to the host VAX-11/780 minicomputer to do the remaining serial calculations required to compute the estimate of the $r(\zeta_k|\theta_k)$.

The MPP implementation of the main portion of the contextual classifier has several advantages over a conventional serial implementation. The obvious advantage is that calculations for 16384 pixels can be performed in parallel. Less obviously, there are further algorithmic advantages to an MPP implementation. The MPP parallel architecture makes it possible to estimate the context function, $G(\theta^p)$, and perform the summation in the decision rule (Equation 5) in one pass through the data. In a serial implementation, the context function $G(\theta^p)$ must be estimated in one pass through a portion of data, and the decision rule must then be evaluated in a second pass. This implementation feature gives a clear efficiency advantage to the MPP implementation. In addition, this feature also gives a subtle accuracy advantage to the MPP implementation since now we can obtain unique estimates of the context function for each pixel. In fact, with the MPP parallel architecture it actually costs less to compute unique values of the context function for each pixel than to compute a block average value of the context function. Because of computation and core memory limitations, a serial implementation is forced to use one average estimate of the context function in classifying a block of data (in Ref. 4 the block sizes ranged from 10-by-10 to 25-by-25 pixels).

Now we describe the MPP implementation of the contextual classifier in more detail. (For a detailed description of the serial implementation see Ref. 4.) Since the MPP consists of an array of 128-by-128 microprocessors, the contextual classification is performed on 128-by-128 pixel portions of multispectral data. To classify an entire data set, 128-by-128 pixel portions of data must be cycled through the program. (These portions of data must overlap by a certain number of pixels determined by the area over which the context function is estimated -- see below.)

Before the program's main classification loop is entered, the class-conditional probabilities, $f(X_k|\theta_k)$, are calculated for each pixel, and an unbiased estimate of the a priori probabilities of each class is made for each pixel. The main classification loop consists of an outside loop over the ground cover classes 'α' and an inside loop over all possible classification vectors θ^p with θ_p='α' (see Equation 5).

Inside the main classification loop, the context function is estimated for the given combination of classes in the context array. A unique estimate of the context function for each pixel is made from an N-by-N square of data centered at each pixel (typically 9 ≤ N

$\leq$ 25). The estimate for pixels on the outer N/2 pixel edge of the array is taken to be zero and no classification is performed for those pixels. Then the product is formed between the context function value at each pixel and the class-conditional probabilities across the context array giving the contribution to the discriminant function for the given combination of classes. The discriminant function for ground cover class 'α' is accumulated by continuing the loop through all possible classification vectors θ^p with θ_p='α'. Once the discriminant functions have been calculated for all ground cover classes, the classification result at each pixel is taken to be the class with the maximum discriminant function at that pixel.

The direct implementation of the contextual classification decision rule (Equation 5) on either a serial (e.g. VAX-11/780) or parallel (e.g. MPP) computer runs into a problem of insufficient exponential range on most computers. For example, and both the MPP and VAX-11/780 computers, the magnitude range of single precision floating point numbers is approximately 0.29e-38 to 1.7e+38. (Due to efficiency considerations and that fact the MPP currently has no double precision floating point implemented, we do not consider double precision floating point numbers here.) With four- nearest-neighbor context (p=5), we see from the previous paragraph that the estimation of the context function, $G(\theta^p)$, involves the multiplication of 5 numbers. Thus Equation 5 requires the multiplication of a total of 10 numbers together. Since each of these numbers must lie in the range 0.0 < 1.0, and, in practice, often lie in the range 0.0 < 1.0e-4, it is easy to underflow the decision rule and be unable to determine a classification for many image pixels. This difficulty is dealt with by evaluating the natural logarithm (LN) of the decision rule rather than the decision rule directly. This trick effectively compresses the exponential range. For example, an exponential range of 1.0e+38 to 1.0e-38 is compressed to the range of numbers +87.5 to -87.5. (This trick does cause a loss of precision, which, however, is of no consequence here.)

Let

$$d_\alpha(X_{ij}) = \sum_{\substack{\theta^p \epsilon \Omega^p, \\ \theta_p=\alpha}} G(\theta^p) \prod_{k=1}^{p} f(X_k|\theta_k) \tag{7}$$

and

$$d'_\alpha(X_{ij}) = LN(d_\alpha(X_{ij})). \tag{8}$$

Maximization of $d_\alpha(X_{ij})$ in Equation 5 (and 7) based on $d'_\alpha(X_{ij})$ is equivalent to maximization based on $d_\alpha(X_{ij})$. Thus, the decision rule becomes:

$d(X_{ij})$ = the action (classification) α which maximizes

$$d'_\alpha(X_{ij}) = LN\left(\sum_{\substack{\theta^p \epsilon \Omega^p, \\ \theta_p=\alpha}} G(\theta^p) \prod_{k=1}^{p} f(X_k|\theta_k) \right). \tag{9}$$

Let

$$F(X_{ij},\theta^p) = LN\left(G(\theta^p) \prod_{k=1}^{p} f(X_k|\theta_k) \right) \tag{10}$$

and

$$M_\alpha(X_{ij}) = \underset{\substack{\theta^p \epsilon \Omega^p \\ \theta_p=\alpha}}{MAX} \Big(F(X_{ij},\theta^p) \Big). \tag{11}$$

Then

$$\begin{aligned}
d'_\alpha(X_{ij}) &= LN\left(\sum_{\substack{\theta^P \varepsilon \Omega^P,\\ \theta_p=\alpha}} EXP[F(X_{ij},\theta^P)]\right) \\
&= LN\left(\sum_{\substack{\theta^P \varepsilon \Omega^P,\\ \theta_p=\alpha}} EXP[F(X_{ij},\theta^P)-M_\alpha(X_{ij})+M_\alpha(X_{ij})]\right) \\
&= LN\left(EXP[M_\alpha(X_{ij})] \sum_{\substack{\theta^P \varepsilon \Omega^P,\\ \theta_p=\alpha}} EXP[F(X_{ij},\theta^P)-M_\alpha(X_{ij})]\right) \\
&= M_\alpha(X_{ij}) + LN\left(\sum_{\substack{\theta^P \varepsilon \Omega^P,\\ \theta_p=\alpha}} EXP[F(X_{ij},\theta^P)-M_\alpha(X_{ij})]\right) \qquad (12)
\end{aligned}$$

Calculating $d'_\alpha(X_{ij})$ in this way insures that at least one term of the sum does not cause underflow, because the exponent of the maximum term, $M_\alpha(X_{ij})$, is never taken. This procedure also makes it less likely that other terms in the sum will underflow since the $F(X_{ij},\theta^P)$ tend to be large **negative** numbers.

Note that Equation 10 can be rewritten as:

$$F(X_{ij},\theta^P) = LN[G(\theta^P)] + \sum_{k=1}^{P} LN[f(X_k|\theta_k)] \qquad (13)$$

When evaluated in this way $F(X_{ij},\theta^P)$, and thus $d'_\alpha(X_{ij})$, do not require any multiplications. All multiplications are replaced by sums of natural logarithms of the terms.

The value of $M_\alpha(X_{ij})$ is not known prior to the start of the summation in Equation 12. Theoretically we could use the maximum value of $F(X_{ij},\theta^P)$ found up to the current term of the sum, and reshuffle the terms of Equation 12 when a new maximum is found. However, the limits of the exponential range on the MPP (approx. 1.0E+-38) make the use of this technique impractical (an implementation "trick" along these lines may still be pursued, however).

The current implementation of the contextual classifier executes a loop over the $\theta^P \varepsilon \Omega^P$ once to identify the value of $M_\alpha(X_{ij})$, and actually evaluates Equation 12 in a second execution of the loop. We have noticed previously in Reference 8, however, that the following decision function produces classifications that closely approximate those produced by the decision function in Equation 12:

$$d(\mathbf{X}_{ij}) = \text{the action } \alpha \text{ which maximizes } M_\alpha(\mathbf{X}_{ij}), \qquad (14a)$$

or in the notation of Equation 9:

$d(\mathbf{X}_{ij})$ = the action α which maximizes for all $\theta^p \varepsilon \Omega^p$ with $\theta_p = \alpha$

$$d'_\alpha(\mathbf{X}_{ij}) = LN\left(G(\theta^p) \prod_{k=1}^{p} f(X_k \mid \theta_k) \right). \qquad (14b)$$

This approximate version of the contextual classifier is also implemented on the MPP. The advantage of approximate version is that the the loop over the $\theta^p \varepsilon \Omega^p$ need be performed only once.

One more implementation comment is relevant here. Running on the MPP host VAX-11/780 minicomputer is the Land Analysis System (LAS), a package of numerous image analysis and manipulation programs. The LAS is implemented under the Transportable Applications Executive (TAE), which is a portable, uniform, user-friendly user interface. Since we eventually want to make the Contextual Classifier available to researchers from a wide range of earth science applications, we have implemented the Contextual Classifier under TAE and made all image and data files conform to LAS standards.

PRELIMINARY CONTEXTUAL CLASSIFICATION RESULTS

We have thus far obtained preliminary contextual classification results on two data sets using the MPP implementation of the contextual classifier. Other results using a VAX-11/780 minicomputer implementation of the contextual classifier are given in References 3 and 4.

The first data set we will discuss is a subset of a Landsat Thematic Mapper image from northern Virginia near the town of Bowling Green. The data set was developed originally for another study (Ref. 9). This area includes Fort A. P. Hill for which there is extensive ground truth data. (However, only 271 pixels of ground truth data have been extracted and registered for accuracy assessment. A more complete extraction of ground truth data from the air photography is being considered.) Being located only 50 miles south of Washington, D. C., the study area was readily accessible for field investigation to the confirm ground truth data.

According to the investigators who originally developed this data set, "the topography of this part of Virginia consists of gently rolling hills with agricultural areas along the flood plains, marsh and swamps in low lying areas adjacent to rivers and streams, and forests in the upland. The Rappahannock River runs across the northern portion of the study area and there are a number of streams that drain into it. The main types of vegetation in the area are deciduous and coniferous trees, marsh and pasture grasses, and an assortment of agricultural crops. The principal agricultural crops grown here are corn, soybean, and wheat" (Ref. 9).

The version of the data set used in our study is described in the original study as the "full resolution combined dates (full comb.)" data set. This data set consists of registered multi-date 30 meter resolution Thematic Mapper data from March 5, 1984; July 29, 1982; and November 2, 1982. Bands 3, 4 and 5 of the March and November data sets were used and bands 3 and 4 of the July data set was used. We did not develop our own multivariate normal model for the ground cover classes in the scene, but instead used the mean vectors and covariance matrices generated by the original study for our class-conditional density functions. These classes were obtained through a supervised technique resulting in covariance matrices with generally much less spread than covariance matrices obtained from the common unsupervised clustering technique for generating the class-conditional density functions.

(This data set was used to shake-down the implementation of the algorithms. We encountered some difficulty in our early implementation of the algorithms due to the fact that the covariance matrices had very little spread. Because of this, the entire data set was not truly represented by the classes chosen and some data points produced low values for all class-conditional density functions. We found that simple thresholding was not satisfactory, and had include normalization steps in the implementation of the unbiased estimator. This was all complicated by the fact that we implemented the algorithms on the NASA/Goddard Massively Parallel Processor which for a time had floating-point math without underflow and overflow detection. We had to wait for an implementation of underflow detection before the algorithm worked properly. Underflow detection may not have been required for covariance matrices with wider spreads.)

For this data set we obtained an overall classification accuracy of 79.7% (216 correct classifications out of 271 test pixels) for the contextual classifier. This compares to an overall classification accuracy of 77.5% (210 correct classifications out of 271 test pixels) for a conventional per-pixel uniform-priors maximum likelihood classification. This conventional classification was obtained using the standard BAYES classification program in the Goddard Land Analysis System (LAS) software package. We evaluated over five ground cover classes: wetlands (and seasonal wetlands), water, barren land, forest and agriculture. The full classification contains 158,105 pixels (roughly 512 by 309 pixels), and was performed in less than one hour (wall clock time) on the MPP.

As mentioned earlier, the ground truth used for deriving the classification accuracy results for this data set consisted of manual ground cover class determinations at 271 pixel locations scattered throughout the data set (see Ref. 9). We feel that a better evaluation of the contextual classifier would be obtained by evaluating the classification results against a more extensive ground truth map. We are pursuing an effort to develop a more extensive ground truth map for the area from aerial photographs that were taken over the same time period when the TM data was gathered.

The next data set that we will discuss in the Anderson River airborne Multispectral Scanner (ABMSS) data set. This data set is a part of a SAR/MSS data set that was acquired, preprocessed, and loaned to us by the Canada Centre for Remote Sensing (CCRS), Department of Energy, Mines, and Resources, of the Government of Canada. This data set covers a 2.8km by 2.8km area in British Columbia, Canada near the Anderson River with terrain elevations ranging from 330 to 1100 meters above sea level. The data were geometrically corrected by CCRS to the Universal Transverse Mercator (UTM) projection at a spatial resolution of 50 meters. A pixel-by-pixel ground cover map was digitized by CCRS from a detailed forest cover map prepared by the staff of the Pacific Forest Research Centre of Canada from aerial photography and more than 20 ground plots (Reference 10).

For this data set we obtained an overall classification accuracy of 81.0% for the contextual classifier. This compares to an overall classification accuracy of 80.5% for the standard BAYES classification program. We evaluated over three ground cover classes: clearcut, hemlock and douglas fir mix. The full data set is 57 pixels by 57 pixels of which the center 49 pixels by 49 pixels were classified by the contextual classifier (a four pixel border was required because of the 9-by-9 pixel window used to estimate the context function). Both the contextual classifier and the BAYES classifier were evaluated over the center 49-by-49 pixel portion of the ground truth data.

We are not happy with the class mean vectors and covariance matrices that we developed for this data set, especially since the original study of this data set obtained an overall accuracy of 88% using per-pixel classification techniques (Ref. 10). This result was obtained for a more difficult discrimination task of classifying into eight ground cover classes: douglas fir, douglas fir mixed with lodgepole pine, douglas fir mixed with cedar, douglas fir mixed with hemlock, hemlock mixed with douglas fir, hemlock mixed with cedar, clearcuts, and bare rock. We have contacted the Principal Investigator for the original study, and have arranged for obtaining the class mean vectors and covariance matrices that were developed for that study. Unfortunately, the publication schedule precludes including results using those class means and covariances in this paper.

CONCLUDING REMARKS

Earlier studies (Refs. 3 and 4) using a VAX-11/780 minicomputer implementation of the contextual classifier obtained classification accuracy improvements of 2% to nearly 6% for small 50-by-50 pixel data sets. These classification runs generally took 3 to 4 hours (wall-clock) to complete. We have implemented the contextual classifier on NASA Goddard's Massively Parallel Processor in order to enable the testing of the contextual classifier on reasonably sized data sets (e.g. 512-by-512 pixels). Preliminary tests have shown that a 512-by-390 pixel data set can be classified with the contextual classifier in approximately one hour (wall-clock) on the MPP. In this implementation of the contextual classifier on the MPP we made no concerted effort to come up with the most efficient implementation possible on the MPP. Still, this relatively inefficient implementation provides better than a 100-fold speed-up over a fairly efficient VAX-11/780 implementation of the algorithm. This amount of speed-up is sufficient to make it possible for the first time to study the effectiveness of this classifier on several different data sets of reasonable size (e.g. 512-by-512 pixels).

The preliminary classification accuracy results reported in this paper for the MPP implementation of the contextual classifier are not as impressive as earlier results obtained from a VAX-11/780 minicomputer implementation of the classifier. Different data sets were used in the earlier study. Also, we expect that our results will improve once certain aforementioned problems are taken care of concerning the data sets used, and once the contextual classifier in run on several other well constructed data sets.

One final note. It makes little sense to compare the speed of the contextual classifier as implemented on a vector supercomputer such as a Cray to the speed of the implementation on the MPP. Devising an implementation on the MPP that effectively uses the parallelism of the MPP is very easy and natural, whereas it would be much more difficult to develop an implementation on a vector supercomputer that effectively exploits that type of parallelism. Being such an easy and natural implementation, the MPP implementation lends itself much more effectively to experimentation with the algorithm.

REFERENCES

1. Williams, D.L., J.R. Irons, B.L. Markham, R.F. Nelson, D.L. Toll, R.S. Latty, and M.L. Stauffer, "A Statistical Evaluation of the Advantages of LANDSAT Thematic Mapper Data in Comparison to Multispectral Scanner Data," **IEEE Trans. on Geoscience and Remote Sensing**, 1984, Vol. GE-22, No. 3, pp. 294-302.

2. Toll, D.L., "Effect of Landsat Thematic Mapper Sensor Parameters on Land Cover Classification," **Remote Sensing of Environment**, 1985, Vol. 17, pp. 129-140.

3. Swain, P.H., S.B. Vardeman, and J.C. Tilton, "Contextual Classification of Multispectal Image Data," **Pattern Recognition**, 1981, Vol. 13, No. 6, pp. 429-441.

4. Tilton, J.C., S.B. Vardeman, and P.H. Swain, "Estimation of Context for Statistical Classification of Multispectral Image Data," **IEEE Trans. on Geoscience and Remote Sensing**, 1982, Vol. GE-20, No. 4, 4, pp. 445-452.

5. Batcher, K.E., "Design of a Massively Parallel Processor," **IEEE Trans. on Computers**, 1980, Vol. C-29, No. 9, pp. 836-840.

6. Schaefer, D.H., J.R. Fischer and K.R. Wallgren, "The Massively Parallel Processor," **AIAA Journal of Guidance, Control, and Dynamics**, 1982, Vol. 5, No. 3, pp. 313-315.

7. Kettig, R.L. and D.A. Landgrebe, "Computer Classification of Remotely Sensed Multispectral Image Data by Extraction and Classification of Homogeneous Objects," **LARS Technical Report 050975**, Laboratory for Applications of Remote Sensing (LARS), Purdue University, West Lafayette, Indiana 47907, May 1975.

8. Tilton, J.C., "Contextual Classification of Multispectral Image Data: Approximate Algorithm," **LARS Technical Report 081580**, Laboratory for Applications of Remote Sensing (LARS), Purdue University, West Lafayette, Indiana 47907, August 1980.

9. M. W. Snyder and M. H. Story, "Classification of Multidate Thematic Mapper Data," **Technical Papers of the 1986 ACSM-ASPRS Annual Convention, Volume 4: Photogrammetry**, Washington, D. C., pp. 360-369, March 16-21, 1986.

10. D. G. Goodenough, B. Guidon, P. M. Teillet, A. Ménard, and J. Zelek, "The CCRS SAR/MSS Anderson River Data Set," presented at the **International Association on Pattern Recognition Technical Committee 7 Workshop**, Paris, France, October 23-24, 1986.

PHYSICS SESSION II

Session Chair: Dr. Richard L. White
Space Telescope Science Institute

C-3

PRECEDING PAGE BLANK NOT FILMED

N87-26552

PLASMA SIMULATION USING THE MASSIVELY PARALLEL PROCESSOR

C. S. Lin, A. L. Thring, J. Koga, and R. W. Janetzke
Department of Space Sciences
Southwest Research Institute
San Antonio, Texas 78284

ABSTRACT

Two-dimensional electrostatic simulation codes using the particle-in-cell model are developed on the Massively Parallel Processor (MPP). The conventional plasma simulation procedure that computes electric fields at particle positions by means of a gridded system is found inefficient on the MPP. The MPP simulation code is thus based on the gridless system in which particles are assigned to processing elements and electric fields are computed directly via Discrete Fourier Transform. Currently the gridless model on the MPP in two dimensions is about nine times slower than the gridded system on the CRAY X-MP without considering I/O time. However, the gridless system on the MPP can be improved by incorporating a faster I/O between the staging memory and Array Unit and a more efficient procedure for taking floating point sums over processing elements. Our initial results suggest that the parallel processors have the potential for performing large scale plasma simulations.

Keywords: plasma simulation, gridded system, gridless system, discrete Fourier transform

INTRODUCTION

Plasma simulations have been used extensively in space physics and fusion energy research. Although the basic physical principles of plasma physics are well understood, the coupled physical phenomena are so complex that analytical studies cannot be easily carried out. With the advent of powerful supercomputers, computer simulations have been used to gain insight into physical phenomena.

One class of plasma simulation model is the particle-in-cell (PIC) model, which treats the plasma at the microscopic level by following the motion of a large number of particles. We have been using a particle-in-cell code to examine the instabilities of natural electron beams observed by the NASA Dynamics Explorer satellite at high altitudes ($> 10,000$ km) (Ref. 1). The simulation code typically uses over 400,000 simulation particles and a 32×32 spatial grid in the two-dimensional problem. It is difficult to improve the spatial resolution with our available resources, because a finer spatial grid would sharply increase the simulated particle number and would require unreasonable computer time. Likewise, three-dimensional simulations are beyond our capability at the present time. Buzbee (Ref. 2) examined the parallel properties of particle codes for fusion studies and concluded that a large percentage of plasma simulation programs can be processed in parallel. We investigated potential improvement in two-dimensional and three-dimensional plasma simulations by carrying out simulations on the Massively Parallel Processor (MPP).

Particle-in-cell simulation models have been developed for the parallel-architecture CHI computer by Dawson et al. (Ref. 3). Their results indicate that the system is cost-effective and offers a significant improvement in performance. The CHI plasma simulation system consists of six microprocessors: one array processor, one macro-processor and four I/O processors. The array processor does most of the calculations, the I/O processors move data around and the macro-processor handles scheduling and control. The parallel processing among these processors accounts for most of the performance advantage of the system. Buzbee (Ref. 2) has implemented a particle-in-cell code on two parallel processing devices–the UNIVAC 1100/84 and the Denelcor Heterogeneous Element Processor (HEP); results from these two parallel processors demonstrated that a large percentage of the total computation can be done in parallel.

For these previous simulation models, the number of parallel processors is less than eight. Plasma simulation models have not been developed for a single-instruction-stream and multiple-data-stream (SIMD) processor like the MPP which has 16,384 arithmetic processors configured as a 128 x 128 array. Without knowing how to take advantage of the unique structure of the MPP, we first attempted to transform our current particle code to run on the MPP. As might be expected, we encountered difficulties in achieving good performance on the MPP using the particle code optimized for the vector computer CRAY. Before

PRECEDING PAGE BLANK NOT FILMED

discussing the difficulties and the new approach, we present some background on the procedures of plasma simulation models.

PARTICLE-IN-CELL MODEL

The particle-in-cell simulation code usually represents the plasma as a large number of computational particles (usually greater than 100,000 particles for each species) which move according to classical mechanics in the self-consistent electromagnetic fields. The two-dimensional spatial system is then divided into fixed spatial cells or grids on which charge densities, potentials, and fields are defined. For the purposes of illustration, we will discuss only the electrostatic model which has no electric current density. In its simplest form, the procedures of the plasma simulation are:

1. computation of the charge density at cell centers. A particle is assumed to have a finite size comparable to the cell size to reduce numerical noise. The charge density of each cell is calculated by accumulating each particle's contribution according to its occupied area in the cell.
2. computation of electric fields from the charge density using Poisson's equation. Poisson's equation is usually solved by using a finite-difference scheme or Fast Fourier Transform.
3. interpolation of the electric forces on the particles from the electric fields at the nearest grid points.
4. application of the electric force to advance the particle velocities and then positions in time using a simple leapfrog scheme.

The mathematical formulas of the electrostatic model are briefly described here. The model consists of finite-size particles, moving in a uniform and constant magnetic field $\vec{B}$ and interacting via a self-consistent electric field $\vec{E}$. The equations of motion are

$$d\vec{v}_i/dt = (q_i/m_i)(\vec{E} + \frac{\vec{v}_i}{c} \times \vec{B}) \qquad (1)$$

$$d\vec{r}_i/dt = \vec{v}_i \qquad (2)$$

Here $\vec{v}_i$, m_i and q_i are the velocity, mass, and charge of the ith particle with the center position denoted by $\vec{r}_i$. The particles have a rectangular shape with a width Δ. The charge density ρ is then

$$\rho(\vec{r}) = \sum_i q_i S(\vec{r} - \vec{r}_i) \qquad (3)$$

The shape function $S(\vec{r} - \vec{r}_i)$ is 1 when both $(x - x_i)$ and $(y - y_i)$ are negative, and is 0 otherwise. Here x and y are the two coordinates of r. Poisson's equation is

$$\nabla^2 \phi = -4\pi\rho(\vec{r}) \qquad (4)$$

where ϕ is the electric potential. The electric field $\vec{E}$ is defined as

$$\vec{E} = -\vec{\nabla}\phi \qquad (5)$$

We will focus on the numerical method of obtaining the electric field $\vec{E}$, which is the key problem in implementing plasma simulation models on parallel computers. The algorithm of solving the electric field $\vec{E}$ from Equations (4) and (5) is usually based on Fourier Transforms. For an infinite continuous system, the Fourier Transforms of Equations (4) and (5) yield

$$\vec{E}(\vec{k}) = -i4\pi\vec{k}S(\vec{k})\rho(\vec{k})/k^2 \qquad (6)$$

where $\vec{E}$, $S(\vec{k})$ and $\rho(\vec{k})$ are the transformed quantities. The infinite continuous transform is defined here as

$$\rho(\vec{k}) = \int d\vec{r}\rho(\vec{r})e^{-i\vec{k}\cdot r} \qquad (7)$$

An infinite continuous system has no grid; therefore, the charge density $\rho(k)$ becomes a summation over the particle positions

$$\rho(\vec{k}) = S(\vec{k})\sum_i q_i e^{-i\vec{k}\cdot r_i} \qquad (8)$$

When the particle number is large, $\rho(\vec{k})$ in Equation (8) is generally too inefficient for computation. Instead a fast algorithm involving a gridded system and Fast Fourier Transform is often used. Figure 1 shows the flow chart of computing electric field $\vec{E}(\vec{r}_i)$ at the location of particles $\vec{r}_i$ in a gridded system. Because the Fast Fourier Transform of $\rho(\vec{k})$ is computed from a small number of $\rho(\vec{r}_g)$ at the cell centers $\vec{r}_g$, the algorithm is very efficient. The saving in computation makes up for the extra steps in collecting cell charge densities and interpolating the electric field at particle positions.

Gridded System On the MPP

Plasma simulation models using grids have been tested on several parallel computers and found to be feasible (Refs. 2-3). Our first plasma simulation model on the MPP therefore uses the gridded system. The program assigns a particle to each processing element in the MPP. Because the Array Unit has 16,384 processing elements, the program updates positions and veloci-

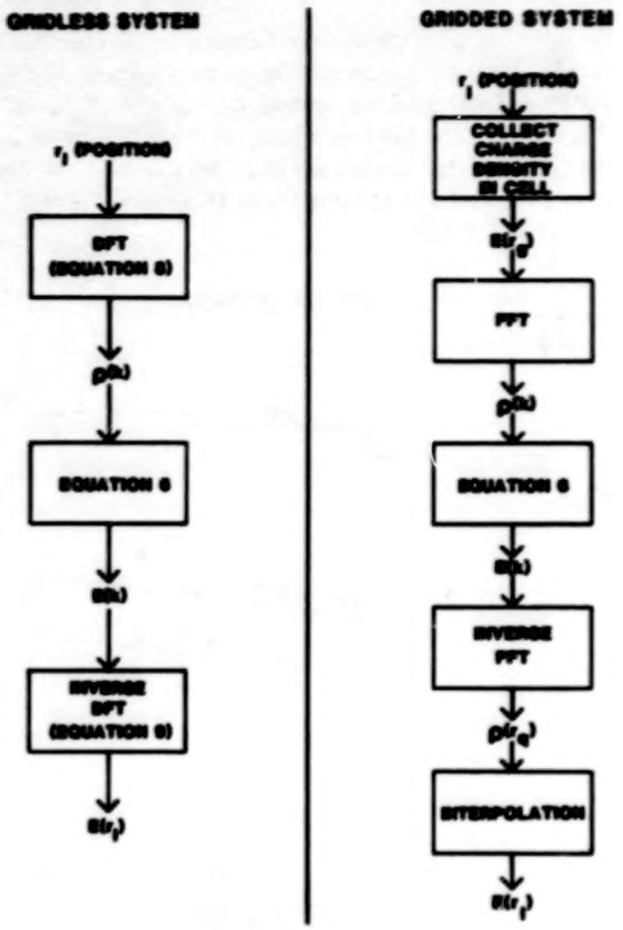


Figure 1. Flow chart of computing electric fields in the gridded and gridless systems.

ties of 16,384 particles at the same time. However, we soon realized that collection of cell charge density and interpolation of electric field at particle positions cannot be computed in parallel or vectorized.

We investigated the possibility of performing serial computations on the VAX, the frontend computer of the MPP. The serial calculations involved in computing charge density at cell centers (Step 1) were completed on the average in 7 seconds on the MPP and 14.5 seconds on the VAX. A separate timing of the number crunching portion on the MPP yields only 1.2 milliseconds, corresponding to a speed of approximately 230 Mflops (millions of floating point operations). Therefore the MPP time of performing Step 1 is essentially I/O time between the staging memory and the VAX. This approach is clearly not acceptable because of the excessive I/O time.

The problem is somewhat alleviated on some supercomputers, which can perform fast serial computation or do indirect indexing (Ref. 4). Some speedup in charge collection on the MPP might be achieved by using an efficient sorting algorithm. However, the interpolation cannot be performed efficiently on the MPP because it has only nearest-neighbor communication and a small memory in the Array Unit. We finally discarded the gridded system for the MPP and investigated the alternative approach described below.

GRIDLESS SYSTEM

The algorithm for the gridless system as shown in Figure 1 is much simpler than the gridded system. Equations (6) and (8) are used directly to compute the Fourier Transform of electric field $\vec{E}(\vec{k})$. No interpolation is needed to obtain the electric field at particle positions. Instead, the electric field at the particle position is computed by using the inverse Discrete Fourier Transform

$$\begin{aligned}\vec{E}(x,y) &= \int_0^\infty dk_x dk_y \vec{E}(k_x,k_y) e^{i(k_x\cdot x+k_y\cdot y)} \\ &= \sum_{n,m} \vec{E}_{nm} e^{i(2\pi/L)(nx+my)} \qquad (9)\end{aligned}$$

Numerical evaluation of the integral in Equation (9) is inefficient. The integral is thus converted to discrete summations by assuming that the system is periodic and has a length of L. The summation is truncated by keeping only the low order terms; we have only kept terms with $n \leq 16$ and $m \leq 16$ in two dimensions. Since the right hand side of Equation (9) depends on only the local parameters (particle position), the calculation of $\vec{E}(x,y)$ can be parallelized.

On the vector computer, the algorithm for the gridless system is much slower than that for the gridded system. Figure 2 shows the timing of the algorithms run on the CRAY X-MP at the San Diego Supercomputer Center. The algorithm of computing electric fields for the gridless and gridded systems in one dimension is shown in Figure 1. For particle number $N = 16384$, the gridless system requires about 15 times more CPU time than the gridded system to obtain comparable results on the CRAY X-MP. The reason for this remarkable difference in speed is simply that the gridless system uses a Discrete Fourier Transform whereas the gridded system uses a Fast Fourier Transform. The number of operations to be performed for the gridless system is proportional to NN_k, where N_k is the number of Fourier modes. For the gridded system the number of operations is proportional to $N_g \log N_g$, where N_g is the number of cells and is usually taken to be 32 for a model. Since N_k is chosen to to be $N_g/2$ and $N \gg N_g$, the gridless system has many more arithmetic operations than the gridded system.

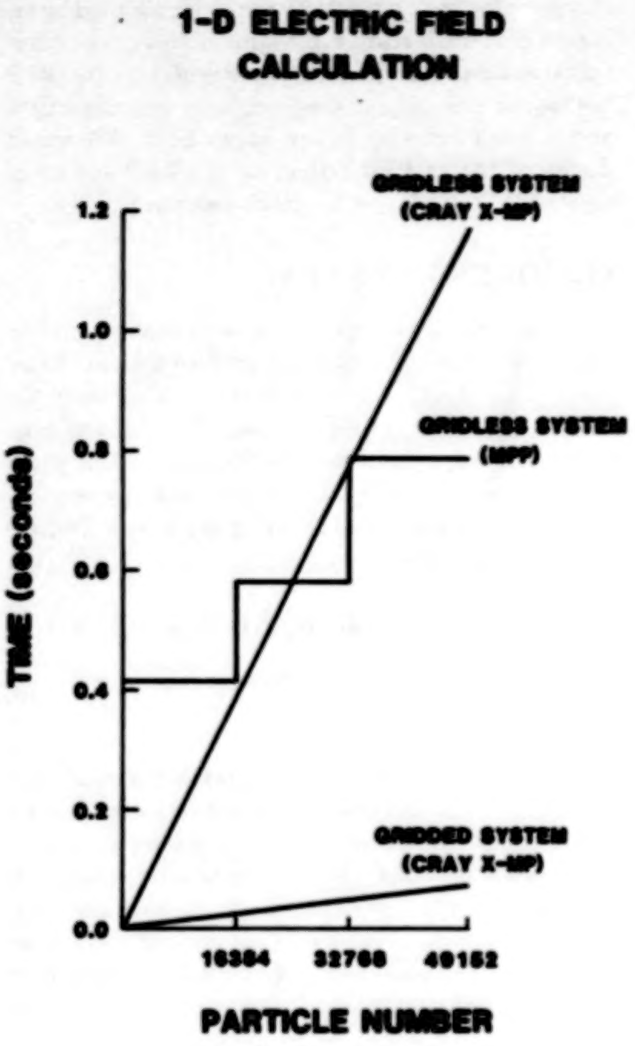


Figure 2. Algorithm timing of electric field calculations in one dimension.

We next discuss the algorithm timing for the gridless system run on the MPP. The MPP CPU time we present includes only processing times of the Array Unit for arithmetic operations and the Main Control Unit for controlling I/O. The I/O time between the staging memory and the Array Unit is not yet included in our algorithm timing because we have not yet optimized the I/O and learned how to determine the I/O time accurately. It appears that the I/O time is many times the MPP CPU time.

In CPU time required to compute electric field in the one-dimensional gridless system, the MPP is comparable to the CRAY X-MP in speed (Figure 2). The MPP is more efficient than the CRAY X-MP for the gridless system because it operates on 16,384 particles simultaneously and thus the number of operations is proportional to $(N/16384)N_k$. Note that the gridless system algorithm is highly parallelized on the MPP.

We next compare the timing of computing electric field in two dimensions between the gridless system on the MPP and the gridded system on the CRAY X-MP (Figure 3). The MPP is about 24 times slower than the CRAY X-MP in the case of 16,384 particles. When $N = 3 \times 16384$, the MPP is about 16 times slower than the CRAY X-MP.

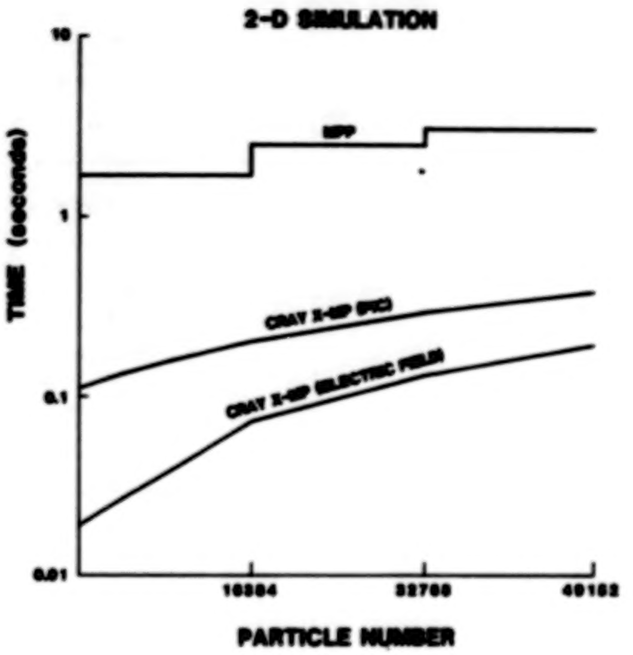


Figure 3. Timing comparison of two-dimensional electrostatic simulation codes between the CRAY X-MP and the MPP. Both the gridded and gridless systems are timed on the CRAY X-MP. Only the gridless system is timed on the MPP.

In breaking down the CPU timing on the MPP, we found that the MPP spends about 60% CPU time for $N = 16384$ and 30% CPU time for $N = 3 \times 16384$ in obtaining the floating point reduction sum, which is a summation taken over 16,384 processing elements. The amount of time spent on reduction sums is excessive since the number of reduction sums is quite small, only 16 for one dimension and 256 for two dimensions. We were able to determine that the routine in the MPP library for summing 32 bit floating point numbers over the processing elements had not been optimized. Specifically, a summation of a 32 bit parallel array took about 2.04 milliseconds, which is about 55 times the CPU time for addition of two 32 bit floating point numbers.

The plasma simulation procedure outlined in the particle-in-cell model includes advancing particle positions and velocities in addition to solving the electric field. Particle positions are checked to determine if particles advance outside the system boundary. The periodic boundary condition is used to adjust the positions of particles advanced outside the boundary

$$x = x - L \quad if\ x > L \tag{10}$$

and

$$y = y - L \quad if\ y > L \tag{11}$$

The procedure of implementing the boundary condition cannot be fully vectorised on the CRAY X-MP. Furthermore, the MPP has an advantage over CRAY X-MP because it can advance 16,384 particles simultaneously instead of 64 particles as processed by CRAY X-MP.

Figure 3 presents the timing for the two-dimensional electrostatic simulation procedure outlined in the particle-in-cell model. The CRAY X-MP uses 0.2 seconds CPU time for simulating a plasma of 16,384 particles, about three times the CPU time needed to compute the electric field. Thus the CRAY X-MP spends about one-third of the CPU time on Steps 1-3 and two-thirds of the CPU time on Step 4. In contrast, the additional CPU time needed for Step 4 on the MPP is negligible (≈ 1 millisecond).

For $N = 16384$, the timing ratio of the two-dimensional electrostatic PIC code is about nine. The ratio remains constant when N increases to 3×16384.

DISCUSSION

The objective of this study is to develop an efficient MPP program to simulate beam plasma interactions in three dimensions. The initial results presented here show some difficulties in reaching this goal. The efficient method typically employed on the CRAY X-MP simulates plasmas in a gridded system, first computing electric fields at the grid points via Fast Fourier Transforms and then interpolating electric fields at particle positions. Because the MPP has only nearest-neighbor communication and a limited memory in the Array Unit, the gridded system is awkward and very slow on the MPP. We have thus adopted a gridless system that assigns a particle to a processing element and computes electric fields at particle positions directly via Discrete Fourier Transform. Currently, the gridless model in two dimensions on the MPP is about nine times slower than the gridded system on the CRAY X-MP. Improvements in the CPU times for the MPP are still possible, since our MPP programs have not been fully optimized. In three-dimensional simulations, the MPP is expected to be much slower than the CRAY X-MP. It is obvious that the speedup factor for the gridless model on the MPP relative to the CRAY X-MP is large since the gridless model is highly parallelized. Indeed, when the performance of the one-dimensional gridless system on both computers is compared, the MPP is as fast as the CRAY X-MP (see Figure 2). However, a much more efficient method using the gridded system can be adapted on the CRAY X-MP but not the MPP.

So far we have only tested the basic ideas of the gridless simulation model and shown that the unique structure of the MPP is suitable for processing this model. We have not yet completed the program for conducting plasma simulations. One critical area unresolved is the transfer of diagnosis quantities from the MPP to the VAX for post processing. In order to minimize the total run time, the program being designed will process I/O between the staging memory and the VAX in parallel with the Array Unit (Figure 4). The

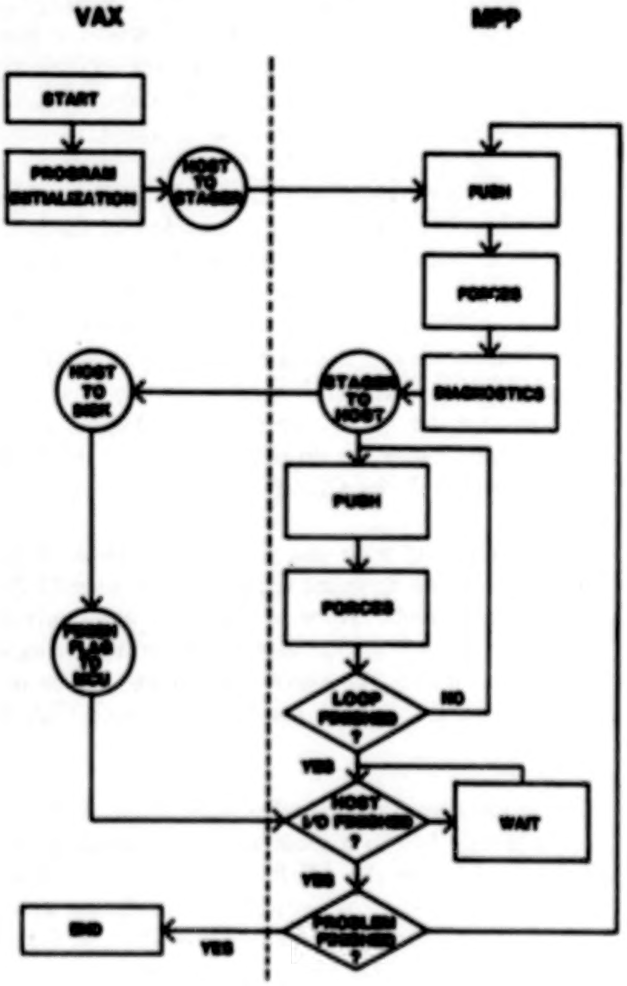


Figure 4. Program design of the plasma simulation code

simulation diagnosis usually needs to examine only a portion of particles for about every hundred time steps or longer. We estimate that the MPP uses about 2.5 seconds to transfer positions and velocities of 16,384 particles. Simulating 16,384 particles in one hundred time steps, the Array Unit will spend about 200 seconds, which is more than enough time to output the diagnosis results. The program will check to see that the I/O is completed before computations continue. These features of our program should ensure that the I/O time between the staging memory and the VAX will not contribute significantly to the total run time.

We have compared only the CPU time between the MPP and the CRAY X-MP. As mentioned before, the I/O time between the staging memory and the Array Unit is not yet timed. Since the I/O is very slow, the MPP performance is misleading without considering the I/O. By simulating only a small number of particles ($N < 49152$), we do not need to use staging memory. Plasma simulation for realistic problems will have many more particles than were used here. Because the Array Unit has a very limited memory (32 floating point words), it is necessary to transfer temporary variables to the staging memory. Our preliminary estimates indicate that tansferring a parallel array between the staging memory and the Array Unit takes from 35 to 200 milliseconds. The I/O time is very slow and highly variable because the I/O initiated by the standard MPP routines is controlled by the front-tend VAX computer. Recently we learned that the I/O time can be reduced by using efficient routines without involving the VAX. But the number of I/O operations for using the staging memory as auxilliary memory for the Array Unit is estimated to be at least 3000 for $N = 3 \times 16384$. Unless the I/O speed is improved significantly, we do not believe that the MPP can compete with the CRAY computers in speed.

Another difficulty with the plasma simulation on the MPP is the floating point reduction sum over 16,384 processing elements. We were unable to investigate an efficient algorithm for computing the reduction sum, but were told that the current algorithm can be improved considerably (private communication with E. Seiler).

Having mentioned the disadvantage of the MPP, we feel it is only fair to discuss some advantages of the gridless model on the MPP. Implementation of the simulation procedure is simplified by assigning a particle to a processing element (see flow chart in Figure 1). Without using the grid, the simulation code also avoids numerical noises due to the grid. Finally, little effort is needed to extend the code to three dimensions. The experience of developing plasma simulation codes on the MPP has proved to be more challenging than anticipated. Our original attempt, implementing the conventional particle-in-cell code on the MPP, was fruitless. Because we assigned particles to processing elements, we encountered the difficulty of indirect addressing in the interpretation of forces at particle posistions from the nearest-neighbor grid points. The MPP currently cannot address indirectly other processing elements. Indirect addressing can be avoided in the gridded system if the cells are mapped to processing elements (Ref. 5).

We have investigated the gridless system approach that uses extensive parallel computation and minimal communication among parallel processors. Our results so far indicate that the MPP, with improvements in both hardware and software, might be acceptable for large scale scientific computation. The needed improvements include a faster I/O between the staging memory and the Array Unit, more memory in the Array Unit, and a more efficient procedure for the reduction sum. With these improvements the gridless system of particle simulations has the potential to be a useful and cost-effective method for simulating plasma phenomena on parallel computers.

ACKNOWLEDGEMENT

We thank E. Seiler of the MPP User Support Office for his valuable assistance in learning to use the MPP. We acknowledge San Diego Supercomputer Center for the use of the CRAY X-MP. We also acknowledge the joint grant of Digital Equipment Corpotion and National Science Foundation for providing a MicroVAX II, which is used for get access to the CRAY and the MPP. This work is supported by Southwest Research Institute Internal Research Program, NSF grant ATM-8405536, and NASA contract No. NAS8-32488.

REFERENCES

1. Lin, C. S., D. Winske, and R. L. Tokar, "Simulation of the Electron Acoustic Instability in the Polar Cusp," J. Geophys. Res., 1985, 90, pp.8269-8280.

2. Buzbee, B. L., "Plasma Simulation and Fusion Calculation," High-Speed Computation, ed. by J. S. Kowalik, Springer-Verlag, 1984, pp.417-424.

3. Dawson, J. M., R. W. Huff, and C. C. Wu, "Plasma Simulation on the UCLA CHI Com-

puter System," (Proceeding of the National Computer Conference) 1978, pp. 395-407.

4. Nishiguchi, A., S. Orii, T. Yabe, "Vector Calculation of Particle Code," Journal of Computational Physics, 1985, 61, pp. 519-522.

5. Gledhill, I. M. A., and L. R. O. Storey, "Particle Simulation of plasmas on the Massively Parallel Processor," (Proceedings of the First Symposium on the Frontier of Massively Parallel Scientific Computation) NASA/Goggard Space Flight Center September 24-25, published in December 1986.

N87-26553

FREE-ELECTRON—LASER SIMULATIONS ON THE MPP

Scott A. Von Laven*, KMS Fusion, Inc., Ann Arbor, Michigan and
Lorie M. Liebrock, Michigan Technological University, Houghton, Michigan

ABSTRACT

Free-electron lasers (FELS) are of interest because they provide high power, high efficiency, and broad tunability. FEL simulations can make efficient use of computers of the MPP class because most of the processing consists of applying a simple equation to a set of identical particles. A test version of the KMS Fusion FEL simulation, which resides mainly in the MPPs host computer and only partially in the MPP, has run successfully.

KEY WORDS: laser simulation, free-electron laser, massively parallel processing

INTRODUCTION

Free-electron lasers (FELS) have demonstrated high power output, high efficiency, and broad tunability from the microwave to the visible spectrum [1]. One-dimensional analyses, in large part, have guided the development of FELS to this point. As experimenters strive to optimize performance, the importance of more detailed analyses is increasing.

An FEL produces radiation when a relativistic beam of electrons passes through a periodic static transverse magnetic field (the wiggler shown in Fig. 1). The electron trajectories are perturbed by the radiation, and the beam becomes bunched on the scale of the radiation wavelength. This leads to coherence and high power. Gain, power at saturation, coherence, bandwidth, and other aspects of laser performance depend on the details of the interaction between the electrons and the fields.

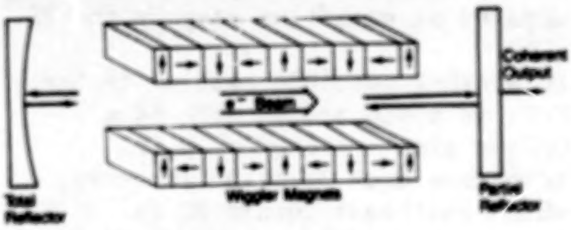


Figure 1. Conceptual FEL

SIMULATION

We simulate the operation of an FEL by following the three dimensional trajectories of a beam of electrons through a wiggler. The inhomogeneous wave equation for a particular mode is used to update the amplitude and phase of that mode; the radiation produced at a number of discrete frequency channels is calculated to determine the gain as a function of laser frequency. The Lorentz force equation is used to update the positions and velocities of the simulation electrons. Updating particle information, according to this simple prescription, accounts for most of the processing time and seems to be a task well suited to the MPP. For serial simulations on the KMS VAX 11/750, we restrict the number of particles to the order of 100. The MPP can accommodate many more particles. Consequently we can study a wider range of beam density

---

*Now at Mission Research Corp., Albuquerque, NM.

PRECEDING PAGE BLANK NOT FILMED

profiles and energy distributions. On the 11/750 we restrict the radiation to a single mode and a few channels. On the MPP we can use more channels and more modes.

In principle one can structure the simulation so as to use a small amount of memory per processing element (PE) and to distribute the computational load evenly across PEs. For the present form of the simulation this is best done by assigning one particle to each PE. The particle information is stored in the local PE memory and is updated at each time step by the PE.

In another possible mapping to the MPP, we could assign each PE a unique electron-channel pair. In this case each 32-by-32 subarray, whose southeast corner PE is directly readable by the master control unit (MCU) of the MPP, represents one frequency channel. Each PE within the subarray is responsible for the spatial coordinates and velocity components of one electron. The same set of 1024 electrons (or multiples of that number) is mapped onto each subarray. The radiative contributions of the electrons to a particular channel are computed and summed within the channel's subarray. The new channel amplitudes and phases are read from the respective corner PEs by the MCU, which can effectively broadcast to all PEs the field information necessary to update the electrons' velocities and positions. More electrons, channels, and modes can be handled by partitioning the simulation into 128-by-128 pieces each with a structure similar to the mappings described above.

As a development strategy we elected to make heavy use of the host-to-MPP call capability. Routines were converted from Fortran to MPP Pascal one at a time starting at the lowest level routines. In this way errors generated during code conversion were more easily isolated and corrected. A disadvantage of this method is that the routine and stager calls must be carefully rewritten for each intermediate version. A simpler software interface between the MPP and its host, a friendlier debugger, and more PE and MCU memory would reduce this disadvantage and permit code development for all users to proceed more smoothly and more quickly.

The structure of the code is shown in Fig. 2. The diagnostics and field pushing routines are called much less frequently than the particle pushing routine (inside circle). The latter currently run in the MPP while the rest of the code runs in the host. A description of the physical model was published earlier (ref. 2).

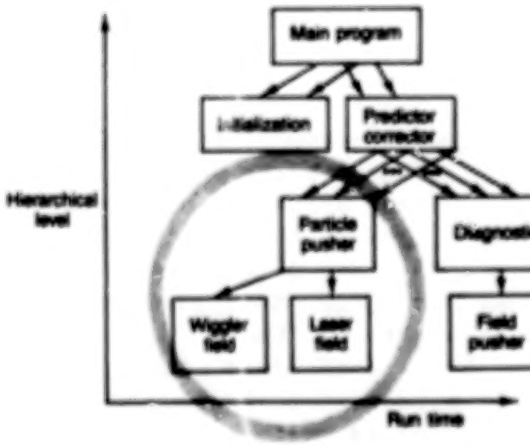


Figure 2. Structure of KMS Fusion 3D FEL simulation

RESULTS

Direct comparisons between serial runs and parallel runs with 16,384 particles have not been performed. Instead, fewer particles were used and the results interpreted accordingly. Until all loops over the particles are in the MPP, even the parallel runs must be performed with a restricted number of particles. Overall processing and elapsed time are the only performance diagnostics used so far. Thus these results are qualitative. For serial runs processing time increases slightly faster than the number of particles. A typical run with 100 particles traversing a one-meter interaction region requires a few minutes of processing time on the host VAX 11/780. Of the runs involving the MPP, those employing the code version represented by Fig. 3 are of the greatest interest. For these runs (with 100 particle or fewer) the processing time is a few minutes and does not increase much with the number of particles. This is because fewer of the loops over the particle index are being executed serially. Versions with only one or two MPP-resident routines required three to five times more processing time.

Only the actual application of the incremental change in particle phase-space coordinates (performed by the predictor-corrector) remains to be parallelized. Then 16,384-particle one-meter runs should still only require a few minutes of processing time. An attempted parallel implementation of the predictor corrector required the use of the stager for parallel-array storage. Ironically, the extra statements required for data swaps pushed the size of our code beyond the capacity of the MCU memory. The generation of more efficient assembly code from MPP pascal may solve this problem. Otherwise we may need to switch to a simpler predictor corrector at the expense of imposing a smaller time step.

ACKNOWLEDGEMENT

Tim Busse provided expert programming assistance throughout this effort.

REFERENCES

1. T.C. Marshall, "Free-Electron Lasers," Macmillan, New York, 1985.

2. H. Takeda, S. Segall, P. Diament, and A. Luccio, "Stable Off-Axis Electron Orbits and Their Radiation Spectrum in a Helical Wiggler," Nucl. Instr. and Methods, A237, 234 (1985).

N87-26554

SIMULATION OF CHARGED PARTICLE TRANSPORT ON THE MPP

James A. Earl
University of Maryland
Department of Physics and Astronomy
College Park, MD 20742

ABSTRACT

Computations of cosmic-ray transport based upon finite-difference methods are afflicted by instabilities, inaccuracies, and artifacts. To avoid these problems, we have developed a Monte Carlo formulation which is closely related not only to the finite-difference formulation, but also to the underlying physics of transport phenomena. Implementations of this approach are currently running on the Massively Parallel Processor at Goddard Space Flight Center, whose enormous computing power overwhelms the poor statistical accuracy that usually limits the use of stochastic methods. In a Monte Carlo simulation of rectilinear transport, the coherent and diffusive effects that appeared are in good quantitative agreement with both finite-difference and analytic calculations.

Keywords: cosmic-rays, interplanetary medium, magnetic fields, particle transport

INTRODUCTION

The diffusion idealization, which has been almost universally invoked in discussions of cosmic-ray transport, is easy to treat analytically. However, many observed phenomena give clear evidence for non-diffusive effects. One example is the so-called "scatter free" propagation of kilovolt solar electrons (Ref. 1), which is inconsistent with diffusion, but which can readily be interpreted in terms of a coherent mode of propagation. This mode is novel, but it is just a manifestation in a dynamic situation of non-diffusive effects similar to those considered in the steady-state by classical transport theory (Ref. 2). Although these effects have been described analytically in References 3 and 4, the theory is very complicated. Consequently, there is a need for reliable numerical computations which bypass these complexities and yield concrete results suitable for comparison with observations. This paper explores such computations based on the well-known Monte Carlo method and compares them to computations based upon finite-difference methods. To limit the number of parameters, and to focus the discussion on computational methods, the present discussion is limited to the case of rectilinear one-dimensional propagation of cosmic-rays along a uniform magnetic guiding field on which are superimposed random fields. This leaves out the effects of adiabatic focusing by non uniform guiding fields and of convective motion of the background medium, which are very important in the interplanetary context, but it includes two other essential aspects of charged particle transport. These are a strong inhibition of transport perpendicular to the guiding field and a pronounced anisotropy of the pitch-angle scattering by random fields. Finally, note that the magnetic fields are visualized as static, which means that the velocities of individual particles are constant, and that there is no interaction among particles in the extremely tenuous distribution of cosmic-rays. This situation differs from those considered by plasma physics, but it is closely analogous to those treated by classical transport theory.

PRECEDING PAGE BLANK NOT FILMED

N87-26554

TRANSPORT EQUATIONS

Under the circumstances outlined above, particle transport is described by

$$\frac{\partial f}{\partial s} + \mu\frac{\partial f}{\partial z} = \frac{\partial}{\partial \mu}\,\psi\,\frac{\partial f}{\partial \mu}, \qquad (1)$$

in which f is particle density in phase space, μ is the cosine of the pitch-angle measured with respect to the guiding field, and z is distance parallel to the guiding field. The variable $s = Vt$, where V is particle velocity, plays the role of a temporal parameter. The Fokker-Planck coefficient of pitch-angle scattering is given by

$$\psi = \frac{(3/2\lambda)}{(2-q)(4-q)}\,(1-\mu^2)\,|\mu|^{q-1}, \qquad (2)$$

where λ is the mean free path, and q is an index that measures the anisotropy of scattering (Ref. 5).

In the discrete formulation, the continuous variables are replaced by a three-dimensional grid whose spacings are Δz, $\Delta\mu$, and $\Delta s = V\Delta t$, and the derivatives appearing in equation (1) are replaced by their finite-difference analogs. These replacements lead to the difference equation

$$\begin{aligned}\Delta f = {} & P_{M+\frac{1}{2}}(f_{M+1,Z} - f_{M,Z}) \\ & + P_{m-\frac{1}{2}}(f_{M,Z} - f_{M-1,Z}) \\ & + \mu_M({}_{M,Z-1} - f_{M,Z}), \qquad (3)\end{aligned}$$

which gives the change in the distribution function during a temporal increment Δs, and which can readily be solved by standard numerical methods. In the Monte Carlo formulation, the random history of a large number of particles is followed under the assumption that the coefficients $P_{M\pm 1/2}$ appearing in equation (3) can be interpreted as probabilities that μ will change by $\pm\Delta\mu$ in each time step. In this formulation, the particles move a distance $\mu\,\Delta S$ in each step. This slight difference in the evolution of z from that described by the finite-difference formulation, has a significant effect that will be discussed below.

In Figure 1, the transition probabilities are plotted as a function of μ for two different assumptions about the anisotropy of scattering. Both curves refer to the same mean free path, $\lambda = 480$. The square symbols describe strongly anisotropic scattering ($q = 1.8$) similar to that occurring in interplanetary space. Evidently, the scattering near $\mu = 0$, which is the boundary between forward ($\mu > 0$) and backward ($\mu < 0$) hemispheres, is much weaker than that occurring within each hemisphere. Consequently, this boundary acts as a physically significant barrier which particles find difficult to penetrate. In contrast, the circular symbols, which refer to isotropic scattering ($q = 1.0$) similar to that occurring in molecular collisions or neutron diffusion, describe relatively weak and nearly uniform scattering with no feature at $\mu = 0$.

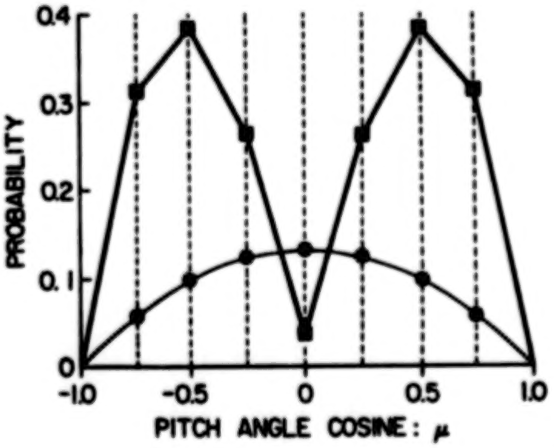


Figure 1. Scattering probabilities for $\lambda = 480$ plotted as a function of pitch-angle cosine. The square points for anisotropic scattering define a curve that is qualitatively different from the one defined for classical isotropic scattering by circular points.

THE ALGORITHM

To implement the Monte Carlo scheme outlined above, each particle was assigned an integer distance and an index corresponding to μ that lies between 0 and 7. Because these parameters occupy only three bytes, there was plenty of storage for several parallel arrays of particles. Consequently, the results given below are based on 31 arrays which contained 31 * 16384 = 507904 particles. Their fate was determined by a single parallel array of random integers (ranging from -32767 to +32767) that was rotated at each temporal step relative to the fixed arrays of particle data, and updated every 31 steps. To implement changes in the pitch-angle cosine, 7 positive integers were chosen in such a way that the probabilities of the current random numbers being larger than this integer are those given by Figure 1. Then the angular index was incremented for those particles whose current random integer was positive and greater than the appropriate probability integer, and decremented for those whose random integer was negative and less than the probability integer with its sign reversed. This approach satisfies the basic requirement that the probabilities of incrementing, decrementing and leaving unchanged the pitch-angle must add to unity. After the pitch-angles had been updated, each particle's distance was changed accordingly. When the desired number of temporal steps had been carried out, particles were binned into an array according to distance and pitch-angle.

ISOTROPIC INJECTION

Results obtained from the MPP for isotropic injection of particles uniformly distributed over pitch-angles are presented in Figure 2 as plots versus distance of the total number of particles in each of 50 bins. This sum over pitch-angles is a measure of the isotropic particle density. Because the total number of particles was large, statistical errors are small and, consequently, are not shown explicitly. However, slight irregularities in some parts of the curves given an indication of their magnitude.

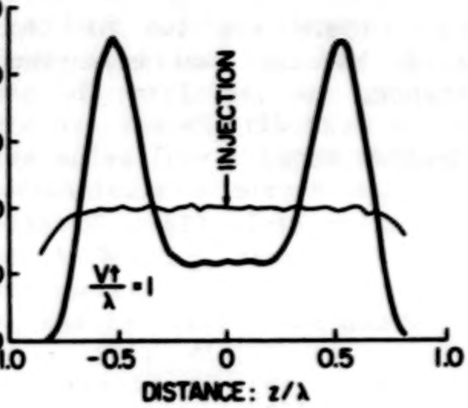


Figure 2. Density profiles shortly after an isotropic injection. The profile for isotropic scattering is featureless, while that for anisotropic scattering exhibits two prominent coherent pulses.

In Figure 2, the darker curve refers to anisotropic scattering (square symbols in Fig. 1), while the lighter one refers to classical isotropic scattering (circular symbols). These density profiles describe a situation very shortly after injection when the particles have had time to move a maximum distance of only one mean free path. The former curve exhibits two peaks, which are completey absent from the latter one. Qualitatively, this manifestation of the coherent mode appears because equal numbers of particles become nearly uniformly distributed in each hemisphere, while very few particles penetrate from one hemisphere to the other through the region of weak scattering near μ = 0. Particles that stay together in each hemisphere, move with nearly the same velocity parallel to the field, but statistical fluctuations of individual velocities give rise to a peak centered around the average displacement. Such features are designated as coherent pulses. There are two of them in Figure 2 because the injection was symmetrical.

THE COHERENT MODE

A detailed analysis (Ref. 3) predicts that the coherent pulses discussed above form a moving Gaussian density profile given by

$$F = \frac{N_o \exp\left\{-\frac{(z-V_* t)^2}{4\pi D_* t}\right\}}{(4\pi D_* t)^{1/2}} , \tag{4}$$

where V_* is a characteristic velocity, which is close to half the particle velocity, and D_* is the coefficient of dispersion, which describes the broadening that arises from statistical fluctuations of individual velocities.

Although the isotropic injection of Figure 2 is the most natural choice of initial conditions, it obscures certain important aspects of the coherent mode. Consequently, to bring out these aspects more clearly, this section will focus on highly collimated injection into a region of anisotropic scattering. More specifically, it will deal with injection of particles in the forward hemisphere at a single value $\mu = 0.6$. Figure 3 shows a density profile obtained under these circumstances, but with all other conditions the same as in Figure 2. Evidently, the profile is very sensitive to initial conditions, for only one coherent pulse appears, and, in place of the second pulse, it is accompanied by a broad feature that is designated as the diffusive wake.

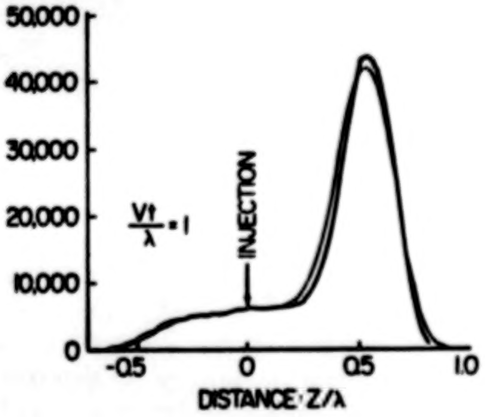


Figure 3. Density profile after a collimated injection, with all other conditions the same as in Figure 2. Evidently, profiles depend sensitively upon conditions at injection. The dotted curve, derived from a finite-difference calculation, is in good, but not perfect agreement with the Monte Carlo results.

The dotted curve gives the result of a finite-difference computation based on Equation 3. The overall agreement is excellent, but there is a slight displacement of the coherent peaks, and the dotted peak is slightly broader than the solid one. The first of these discrepancies is a trivial artifact arising from imperfections in the binning, but the second arises from an important difference between the two computations. More specifically, the finite-difference implementation leads to a dispersive effect which can be described by

$$D_{\#} = \frac{1}{4}V(\Delta z - \frac{1}{2}\Delta s) , \tag{5}$$

and the total dispersion is a superposition of the physical effect described by D_* and the numerical artifact described by $D_{\#}$. In the present example, the condition that ensures that physics dominates, $D_* > D_{\#}$, is met, for $D_* = 4D_{\#}$, but the numerical dispersion is large enough to account for the slight discrepancy between the peaks in Figure 3. To confirm this interpretation, a finite-difference calculation was performed with $D_* = D_{\#}$, and the expected further broadening of the coherent pulse was seen. These details are significant, for they illustrate that the Monte

Carlo method, by virtue of the difference mentioned above in the evolution of z, is relatively free of the artifacts which plague finite-difference calculations. (See Ref. 6 for a discussion of published work that has been affected by such problems.) Numerical dispersion is an especially insidious artifact, for it leads to results that seem plausible, but are quantitatively in error.

ANGULAR DISTRIBUTIONS

To bring out additional features of the coherent mode, Figure 4 shows further results from the anisotropic injection that was discussed above. Here, particle density has been averaged separately over the forward and backward hemispheres. A comparison of the two curves reveals that particles in the pulse are overwhelmingly collimated in the forward direction while those in the wake are predominantly collimated backward. This behavior suggests that the wake can be interpreted as particles scattered out of the pulse which subsequently move coherently away from it in the backward direction.

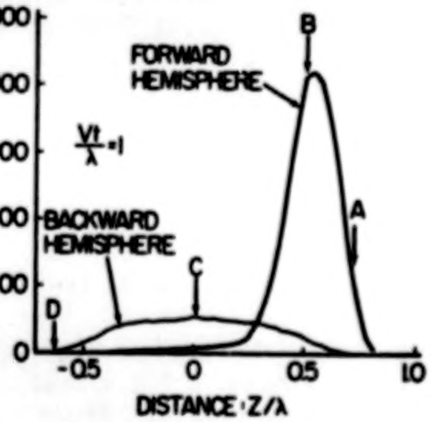


Figure 4. Densities in the forward and backward hemispheres under the same conditions as in Figure 3. The coherent pulse contains particles moving forward. It is accompanied by a broad wake of particles moving backward.

Detailed angular distributions tabulated at four points indicated on Figure 4, and normalized to the same total number of particles, are presented in Figure 5. The distribution at the peak of the pulse (Curve B, circular points) is essentially a mirror image of the one at the center of the wake (Curve C, square points). Both describe near isotropy in one hemisphere associated with nearly zero intensity in the other. This distribution, which is characteristic of the coherent mode, adds weight to the interpretation outlined above.

On the fringes of the density profile, at points A and D, angular distributions (triangular points) are highly collimated in the forward and backward directions, respectively. In the interplanetary context, the fringes would correspond to the the very first particles to arrive in a solar event, which are particularly difficult to describe theoretically, but which are very important in connection with accurate timing of events on the sun. On the MPP, Monte Carlo methods, can give a useful description of this phase of solar events, but statistical fluctuations limit the applicability of the method on less powerful computers.

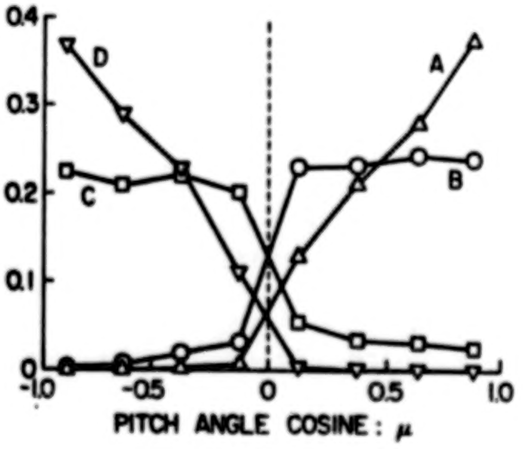


Figure 5. Normalized angular distributions at locations indicated in Figure 4. These distributions exhibit a remarkable mirror symmetry.

THE DIFFUSION LIMIT

Eventually, scattering destroys the strong anisotropies of the coherent mode, and the featureless wake becomes dominant. In this regime, where the familiar theory of diffusion is applicable, the density profile is described by a Gaussian whose width is controlled by the coefficient of diffusion $D = \lambda V/3$ (Ref. 7). Figure 6 shows a profile computed by the MPP in this regime for $Vt = 64 * \lambda$ (solid curve). Evidently, this profile is in very good agreement with the dotted curve which is based upon diffusion theory.

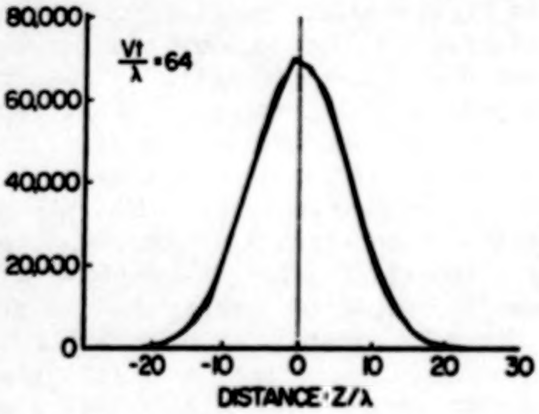


Figure 6. Density profiles in the diffusion regime. Monte Carlo results are in very good agreement with the dotted Gaussian derived from diffusion theory.

CONCLUSIONS

Results obtained from the MPP with the aid of Monte Carlo methods are equivalent in every detail to those based upon careful use of more traditional methods, but they are less subject to error and are closer to the physics. These characteristics offer tremendous advantages in the investigation of exotic transport regimes for which no theoretical description is available. In particular, the formulation of problems in which particles gain or lose energy leads to prohibitively large conventional computations, but their Monte Carlo versions are not significantly more complicated than the one described here. We intend to exploit these advantages in the investigation of two such problems: Adiabatic deceleration of cosmic-rays due to expansion of the solar-wind, and the loss of energy by electrons in radio sources due to synchrotron radiation.

ACKNOWLEDGEMENTS

This research was supported by NASA under Grant NGR-21-002-066.

REFERENCES

1. Lin, R.P., "Emission and Propagation of ~40 keV Solar Flare Electrons", Solar Physics, 15, 453, 1970.

2. Case, K.M. and Zweifel, P.F., "Linear Transport Theory", Addison-Wesley, Reading, MA, 1967.

3. Earl, J.A., "Coherent Propagation of Charged Particle Bunches in Random Magnetic Fields", Astrophysical Journal, 188, 379, 1974.

4. Earl, J.A., "Analytical Description of Charged Particle Transport Along Arbitrary Guiding Field Configurations", Astrophysical Journal, 251, 739, 1981.

5. Jokipii, J.R., "Cosmic-Ray Propagation I, Charged Particles in a Random Magnetic Field", Astrophysical Journal, 146, 480, 1966.

6. Kota, J., Merenyi, E., Jokipii, J.R., Gombosi, T.I. and Owens, A.J., "A Numerical Study of the Pitch-Angle Scattering of Cosmic-Rays", Astrophysical Journal, 254, 398, 1982.

7. Earl, J.A., "The Diffusive Idealization of Charged Particle Transport in Random Magnetic Fields", Astrophysical Journal, 193, 231, 1974.

INVITED PAPER

Prof. Dennis Parkinson
DAP Support Unit
Queen Mary College
London, England

N87-26555

EXPERIENCE IN HIGHLY PARALLEL PROCESSING USING DAP

D. Parkinson
DAP Support Unit
Queen Mary College
University of London
London, England

ABSTRACT

DAP systems have been in day to day use for 10 years and a large amount of user experience has been gained. The profile of user applications is similar to that of the MPP working group.

Experience has shown that contrary to expectations, highly parallel systems provide excellent performance on so-called "dirty problems" such as the 'physics' part of meteorological codes. The reasons for this observation are discussed. The arguments against replacing bit processors with floating point processors are also discussed.

Keywords: DAP, bit serial processors, FORTRAN, balance.

INTRODUCTION

The DAP [2] is generally similar in concept to MPP but differs in some important ways. Probably the most ways in which the DAP differs from MPP are:

a) Memory size : The system view of DAP has always been that of a memory module with processing capability and so emphasis has always been placed upon having the largest memory it was practical to use when the systems were designed. Thus the 64x64 DAP when delivered to QMC had 4K bits per PE, but this was enlarged to 16K bits per PE when the appropriate static RAMs became available at reasonable prices. User experience has shown that many applications are very memory hungry and there is little limit to the memory which users can exploit. Plans for DAP-3 have a minimum memory of 64K bits per PE rising through 256K bits per PE to a current maximum of 1Mbit/PE giving a potential total of 128M bytes of a 32x32 DAP-3 or $\frac{1}{2}$G bytes on a 64x64 system.

b) Interconnection Network : The DAP DAP has a 2D NEWS network similar to that of the MPP but additionally has a second layer network of row and column highways. These highways provide facilities to rapidly fetch and broadcast data. The highways are connected to edge registers in the master control unit and permit data to be selected from any set of processors, one per row/column or broadcast to any of the processors in the row/column. As the same edge registers are used for both row and column highways, 'corner turning' is also supported. The highways are not 'buses' but are essentially multi-way OR gates and so give the DAP the properties of an 'associative processor' [3].

c) Software Systems : An extended FORTRAN dialect is the principal programming language for DAP use [4]. This language supports array processing in data sets with either N or N^2 items on an N*N DAP. Reals with 3, 4, 5, 6 or 7 and 8 byte precisions are fully supported, also available are 1-8 byte integers and bit variables (logicals). The flexibility and efficiency of the DAP-FORTRAN language is such that users very rarely need to use the assembler language APAL. Indeed, the use of APAL is actively discouraged and is only justified for

PRECEDING PAGE BLANK NOT FILMED

specialist bit manipulation algorithms arising in such areas as low precision image processing and customised-format arithmetic routines - floating point users never need APAL.

Above the high level system language DAP-FORTRAN there exists a subroutine library structured on the NAG line containing a wide variety of useful routines. The most widely used routines include Random Number generators, FFT's, tridiagonal equation solvers, sorting routines and a number of general utilities for data re-arrangements. The performance of the random number routines is particularly spectacular [5].

USER EXPERIENCE

The DAP has been used for a wide range of applications in the scientific field. The range of topics is almost identical to that covered at this conference, so there is little point in detailing it here. The application area most represented is Monte Carlo Simulations in such areas as QCD, Ising models, molecular dynamics etc., much to the surprise of many so-called experts who felt that SIMD architectures were unsuitable for Monte Carlo work. In addition to scientific (floating point) work, the DAP has been used for a number of non-numeric problems including:

- searching for large primes
- textual data abstraction
- information retrieval, and
- graphics - in particular, molecular graphics.

The bit serial nature of the processing elements has helped greatly in providing the flexibility to support more than just specialist number crunching.

"DIRTY PROBLEMS ARE NICE!"

The reaction of users to the DAP has been very favourable especially to reprogramming exercise. The expressive power of DAP-FORTRAN has allowed great reductions in program size and complexity, and much higher performance than expected for so-called "dirty problems", i.e. problems requiring many data dependent decisions.

There is a natural tendency to assume that problems with many decisions are not optimum for the DAP/MPP type of computers as the natural way to implement them is by parallel conditional expressions of the type

```
where (temperature<freezing point)
    Heat:=heat + latent_heat
```

In DAP-FORTRAN this would be coded as

```
HEAT(TEMPERATURE.LT.FREEZING POINT)
= HEAT + LATENT_HEAT
```

the array indexing type is called MASK-INDEXING.

As only some processors will update their local values of the variables, the natural tendency is to assume that such code will demonstrate a lower speed-up over 'conventional' computers than, say, unconditional code. Our experience is that the reverse is the case and tasks of the above nature demonstrate better speed-up than the clean unconditional code!

The somewhat paradoxical result is explained when one realises that one is comparing the performance of two machines rather than measuring some absolute computing performance, and what one is really observing is that arrays of bit-serial processors are better suited to "dirty code" than are 'conventional' word based architectures. The explanation is simple but difficult for many people to grasp. Bit-serial systems are primarily optimal for dealing with arrays of logical operations, as these are essential single bit operations. Arithmetic is software produced out of lots of logical primitives, hence the worst performance of the system comes from pure floating point code! The greater the proportion of a problem that is purely logical, the greater the potential is for an array of bit-serial processors to outshine architectures based on floating point pipelines - which try to avoid conditionals.

SYSTEM BALANCE

Some users of DAP have argued that the system would be improved if the bit-serial processors were replaced by floating point units and it is not surprising to me to find a number of the MPP group making the same observation. Care should be taken in evaluating what is really being stated. There are three arguments which can be made against such changes.

1) 'Dirty problems' will really become dirty as the arguments in the previous section become invalid.
2) The parallelism in systems of equal cost becomes much less. The cost (in say mm^2 of silicon) of a 64 bit floating point processor is equivalent to the cost of more than 128 single bit PE's, hence one should realise that the option one is really comparing is that of say, 16K, 1 bit processors and 128 - 64 bit processors.
3) The inter-processor communication rate should be somehow in "balance" with the processor computational capabilities. I have attempted to put some flesh on this concept of balance [6], which I summarise here.

For many parallel algorithms - especially those with results producing global properties of arrays, the computation is carried out in a loop structure of the form:

```
K:=1;
WHILE K<N begin
  move some data a "distance" K;
  perform a computation;
  K:=K*2
  END WHILE
```

typical of such loops are - SUM, Product, Maximum, Minimum, FFT, Sorting etc.

The loop is traversed $\log_2 N$ times

so the cost is:

$(\log_2 N)$ cost of a computational step +

cost of shifts 'distance' 1+2+4+8.... On a mesh connected processor with edge size n and dimensionality d and

$N=n^d$ (for MPP n = 128, d = 2), the cost of such shifts become $d(n-1)*$ cost of moving a data item from one processor to its neighbout.

The Balance Factor can be defined by:

$$B = \frac{\text{time for computational component}}{\text{time for data movement component}}$$

if B = 1 the two costs are equal, i.e. the system is in balance. If B>>1, the computational time is dominating and we would gain in performance if more powerful processors were used. If B<<1, data movement costs dominate and little benefit would appear if the processors were speeded up without improving the data transmission speed.

G = 1/B could be called the Granuality factor of the system as it gives an idea of the minimum recommended number of computational operations that must be performed per data transmission step.

Computation of balance factors for systems such as MPP and DAP suggests that for floating point operations the systems are in approximate balance and no benefit would be gained by using floating point PE's unless the interprocessor network were improved by multibit highways and/or higher dimensionality meshes.

The bit serial processors approach seems to be an optimal route to providing very highly parallel systems which can support a large range of application areas.

Although one could try and produce the computer with the worlds largest MFLOP rating by building arrays of

2^{16} floating point chips, there is little evidence that such machines would produce even 0.1% of their theoretical performance on anything other than highly artificial problems such as the Mandelbrot Set.

REFERENCES

1. Flanders, P.M., Hunt, D.J., Reddaway, S.F. and Parkinson, D, "Efficient High Speed Computing with the Distributed Array Processor", (High Speed Computer and Algorithm Organisation) Academic Press 1977.

2. Parkinson, D. "The Distributed Array Processor" (Computer Physics Comm. Vol.28, pp. 335,) 1983.

3. Foster, C.C. "Content Addressable Parallel Processors", (Van Nostrand), 1976.

4. International Computers Ltd., Technical Manual TP 6918,"DAP-FORTRAN" (see Ref. 2 for brief description)

5. Smith, K.A., Reddaway, S.F., Scott, D.M., "Very High Performance Pseudo-Random Number Generation on DAP", (Vector and Parallel Processors), North Holland, 1985.

6. Parkinson. D, "Organisational Aspects of Using Parallel Computers" (International Conference on Vector and Parallel Computing, Loen, Norway, June 2-6, 1986). To be published.

COMPUTER GRAPHICS SESSION

Session Chair: Prof. John A. Barnden
Indiana University

N87-26556

RAY TRACING ON THE MPP

John E. Dorband
Image Analysis Facility/635
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

Generating graphics to faithfully represent information can be a computationally intensive task. We will present a way of using the MPP to generate images by ray tracing. This technique uses sort computation, a method of performing generalized routing interspersed with computation on a single-instruction-multiple-data (SIMD) computer.

Keywords: Ray Tracing, Graphic Generation, Sort Computation, SIMD, MPP.

INTRODUCTION

New ways of representing data must be developed so that researchers can most effectively focus their time and effort in exploring the ever increasing volume and complexity of data that they acquire and generate.

Since computers are not yet able to discover knowledge for which no description has been given, we must rely on the cognitive abilities of the researcher to recognize undiscovered characteristics of the data. Tools are needed that extend the senses of the researcher deeper into the data he is studying. Just as a telescope allows an astronomer to visually search deep into the universe, so a computer allows a researcher to visually study data in regions that could not be explored through mere imagination.

Computer-assisted perception is one of the most valuable aspects of man-machine interaction, but the amount of information that needs to be transferred between man and machine for human perception to be effectively extended is huge. The visual interface between man and machine has the greatest potential for satisfying this requirement, but a fast and efficient means of converting logical information into visual information needs to be developed. Faster and less expensive computer hardware, along with more efficient algorithms, are needed to place this capability within the reach of most researchers.

The most versatile and realistic means of generating images of objects in three-dimensional space is ray tracing. Ray tracing generates images by simulating the movement of light rays in the three-dimensional space containing the objects to be displayed. Ray tracing is, however, computationally intensive making it both slow and expensive when the three-dimensional space holds many complex objects and especially when animated sequences composed of many images must be generated.

An experimental computer architecture termed single-instruction-multiple-data (SIMD) is being investigated at NASA's Goddard Space Flight Center. SIMD architectures show promise of delivering enormous computational power at less cost than other existing architectures. The Goddard prototype, the Massively Parallel Processor (MPP), has a computational element consisting of a 128 by 128 array of small computers.

It had been recognized that ray tracing, which requires the use of irregular data arrangements, could not be performed easily on the SIMD architecture. This has changed with the recent development of a technique called sort computation (Ref. 1), which uses the regular SIMD computer architecture to process irregular data arrangements more efficiently than was previously thought possible. With sort computation, the MPP can be used to start developing ray tracing as a method of rendering images of objects placed in three-dimensional space.

Sort computation is standard sorting with a twist provided by an enhancement to the usual comparison step in the sort. The enhancement performs computation during the sort. The kinds of computation that can be performed include aggregation and distribution type operations on sets of data records whose key values are the same. The routing mechanism of the sort guarantees that the right data will be brought together at the right time

PRECEDING PAGE BLANK NOT FILMED

to accomplish the computation without the programmer's a priori knowledge of where the data was before the computation or where it will be after the computation. This makes it easy to process irregularly arranged data.

Our initial step toward implementing ray tracing on the MPP was the development of a utility that allows a researcher to view a topographical map in three-dimensional space from any perspective. The map is a 512 by 512 grid of elevation points. These points are initially plotted in three-dimensional space. Then the intersection of the viewing screen with light rays from the view point to each point in the map is calculated. It is then known where all points will be seen on the viewing screen. A sort computation operation, sort minimum, is then used to determine which points are not hidden by other points. Sort minimum finds the closest point on the elevation map to the view point for each position in the viewing screen. Finally, the brightness values of those points that are not hidden are copied to their proper position in the viewing screen, using sort distribution.

This utility, run on the MPP, takes 3.5 seconds to generate an image of 262,144 elevation points. It was implemented on the MPP by a Code 635 summer student. Jennifer Trainor (see Figure 1, Color Plate IV), and is being used to view elevation maps generated from Shuttle Imaging Radar-B (SIR-B) images by Dr. James Strong (Code 636).

SORT COMPUTATION

To truly understand how ray tracing is accomplished on the MPP one must understand sort computation. Computation in general requires both the ability to manipulate elements of data based on the values of other elements of data and the ability to route these elements of data to places where they can affect each other. Sort computation uses sorting as a routing mechanism to support interspersed routing and data manipulation. Sort aggregation and sort distribution are the two basic forms of sort computation. Sort computation is performed on sets of records, grouped according to a key contained in each record. Groups of records contain only records that have been determined by some function to be equal.

Sort aggregation generates an accumulative result for each group of records and places this result in one of the records. Usually it is placed in the first record or the record with the smallest key value. The accumulation operation can be any operation that is both associative and commutative. That is any operation for which the result is not dependent on the order of the operand or the order in which the operations will be performed. Addition, multiplication, and, or, and exclusive-or are examples of valid operations.

Sort distribution copies the value of a field of a specific record in a group into that field in every record of that group. The record that contains the value to be distributed contains a flag that is set to true, indicating that this record contains the value that is to be copied into all other records of that group. Note that there may be more than one record in a group for which the flag is set, as long as they contain the same value in the field that is to be distributed.

Sort computation is really quite simple. First, we must view a sorting algorithm as having two parts, the comparison of records and the routing of records. The comparison of records results in a determination of which of two records being compared is larger. The routing of records takes the result of the comparison and determines where each of the records are to go for their next comparison. Thus, the sort can be viewed as a routing routine and a comparison routine, where the routing routine calls the comparison routine when necessary. All sorts consist of these two parts. Sort computation can use the routing part of any sort algorithm. The comparison routine is replaced depending on what type of computation is to be performed.

The routing routine only determines the order in which the record will end after the sort is through, not how they will be modified. The comparison routine contains all the code that makes any sort computation different from another in terms of how the contents of the records are changed. The comparison routine has two functions. One function is to determine if the two records being compared are in the same group, generally whether or not their keys are equal, and whether a record from one group will come before or after a record from another group. The other function is to modify the records if they both belong to the same group.

SORT AGGREGATION

Pseudo code will be used to describe the following algorithms . The expression "A[5].(B,C,D)" will define an array of 5 records, where each record has 3 fields, B, C, and D.

For the sake of simplicity we will just show that sort summing works. Note that the addition operation can be replaced by any other valid aggregate operation. The command "SORT(SUM,A)" performs the sort sum over the array A defined by "A[n].(K,V)".

```
boolean function SUM(A1,A2)
        given A1.(K,V)
        given A2.(K,V)
        if A1.K = A2.K then
                A1.V = A1.V + A2.V
                A2.V = 0
                return(true)
                end if
        if A1.K < A2.K then
                return(true)
                end if
        if A1.K > A2.K then
                return(false)
                end if
end function
```

Figure 1. Sum routine

SUM (Figure 1) is the comparison routine which will, when used in conjunction with SORT, sum all the values in field V of the records for which the K fields are equal. SUM returns a value true if the records A1 and A2 are in the correct order and false if they are not. SUM will put the sum of all the V fields of records of the same group in the first (or smallest) record in the group.

The proof that this will work as described goes as follows. Even though the keys of the records we are comparing may be equal, SUM can affect their ordering by returning to the routing routine the response that either the records are in the correct order (true) or not (false). This in effect gives order within a group. SUM always designates the record that contains the result of the sum as the smaller of the two records and the larger of the two records contains a value of zero. This means that the sum of the value fields of the records of a group will be contained in the record that was designated smaller than all other records of that group. Let us assume, however, that not all values of records in a specific group were summed into the same record. This means that at least two records contain only part of the result for that group. Each one of these records would have been designated less than all records of that group to which it was compared. Yet, the records which contained partial results must not have been compared to any one of the others or the partial results would have been summed into one of the two records. Thus, each one of the records would have been designated the smallest record of the group. Since there is only one smallest record of a group, there can only be one record that contains the result for any group.

A routine corresponding to SUM can be writen for any operation that is both associative and commutative, as described above.

```
boolean function COPY(A1,A2)
        given A1.(K,F,V)
        given A2.(K,F,V)
        if A1.K = A2.K then
                if A2.F then
                        A1.V =  A2.V
                        A1.V = true
                        return(true)
                        end if
                if A1.F then
                        A2.V =  A1.V
                        A2.V = true
                        return(true)
                        end if
        else
                return(true)
                end if
        if A1.K < A2.K then
                return(true)
                end if
        if A1.K > A2.K then
                return(false)
                end if
end function
```

Figure 2. Copy routine

SORT DISTRIBUTION

Sort distribution is slightly more complex than sort aggregation. The idea in sort distribution is to copy the value of a record in a group of records, which has been flagged as having a valid value for that group, to all records that do not already have that value. The command to perform this would be "SORT(COPY,A)", where SORT is the routing routine, COPY is the comparison routine, and A is the array of records. This array of n records is of the form "A(n).(K,F,V)", where K is the key, F is the valid value flag, and V is the value field. COPY used in conjunction with SORT distributes the correct value for each group to all members of the group (see Figure 2). Like SUM, COPY returns a value true if the records A1 and A2 are in the correct order and false if they are not. COPY puts the same value in all records of the same group or no value at all if no record of the group had its valid value flag set prior to performing the distribution.

The proof that distribution works is similar to that of aggregation. Note that when two records are determined to be in the same group and one of the records contains a valid value, it is copied to the other record and its valid value flag is set. This in effect cause the record with a valid value to be considered both larger and smaller than a record that does not have a valid value. Thus, at the completion of the sort computation, at least the largest and smallest record of each group that had a record with a valid value, will contain a valid value. Assume that a record without a valid value remained so after the sort was complete. If it was either the largest or the smallest record of the group, then there must not have been a record in the group that had a valid value. If it was not the smallest or the largest value of the group, either there was no record in the group with a valid value or it was not compared to a record in the group with a valid value. If there is a record without a valid value and one with a valid value in the same group, there is such a pair that is logically next to each other that has never been compared. If such a pair exists, there is no way of knowing which one is larger, since they have never been compared. Thus, the sort must not have been complete. Therefore, a record can only be left without a valid value after the sort is complete if there were no records in its group with a valid value.

RAY TRACING BY PARALLEL RECURSIVE SUB-DIVIDING OF SPACE

Originally, ray-tracing algorithms were based on an approach that compared each object with each ray to determine whether the ray intersected the object. More recently, the viewing space has been divided into varying size regions depending on how many objects are in the region (Ref. 2-6). If the region has too many objects in it, it is divided into smaller regions. The rays are traced through the regions until they intersect an object. This subdividing of space is done prior to the actual ray-tracing. The collection (a hierarchy) of regions is stored as a structure to be searched during the actual ray tracing.

We recursively subdivide space during ray tracing; therefore we do not have to either store or search the hierarchy of subdivided regions. We also do not actually compare rays to objects to find their intersection, but determine the intersection of subdivided regions (cells) and rays and the intersections of cells and objects. When divided each cell is split into eight subcells of equal size. If a cell does not both intersect a ray and an object it is deleted. Thus, all further processing associated with that cell is terminated. As each remaining cell is subdivided into smaller cells, the only cells that remain are those that intersect at least one ray and one object. The cells are subdivided until the cross-section of the subcell is smaller than some specified fraction of the area that can be viewed through a pixel of the viewing screen.

The position of the intersection of the object and the ray can either be taken as the center of the cell that they both intersect or a more accurate determination of the intersection can be calculated from the analytic description of the object and the ray. Once the intersection of the ray and the object has been determined, reflected or refracted rays may be calculated if further generations of rays are to be simulated, or a brightness value for the ray can be calculated. All ray/object intersections of a generation of rays are calculated before any intersection of the next generation.

This is a highly parallel algorithm since there is no interaction between processing of neighboring cells. The only global processing required is the determination and deletion of all cells that do not intersect both a ray and an object.

The algorithm initially starts with a set of rays and a set of objects. This set of rays usually start at a view point and each ray intersects a simulated viewing screen at different pixel positions.

For the sake of this discussion, we will assume that all computation can be contained within the processor array. In reality data must be swapped to and from the staging memory.

We start with an initial cell that contains all the objects. Each processor holds the description of a ray and an object that have no relation to each other. First it is determined which rays intersect the cell. These rays are then sorted to a region of the array of processors. The remaining rays are considered to be deleted. The same thing is done with the objects. This makes more efficient use of the processors when the ray descriptions are duplicated and moved to processors not being used. At this point there will probably be both rays and objects left, so there will be no reason to delete the only cell.

Each ray and object will be duplicated eight times, moved to different processors, and paired up with one of the eight subcells of the initial cell. For each ray/cell pair, it will be determined if the ray and the cell intersect. If they do not, then the ray/cell pair will deleted. The same will be done with object/cell pairs.

Deletion of cells that do not intersect both a ray and an object can be handled by the application of sort-computation. This is done by creating two types of records, ray/cell and object/cell, that will be sorted together using a sort distribution operation. Each record will consist of a key field and two flag fields, the ray and the object flag. The key field contains the ID of the cell that intersects the ray or the object. The ray flag is true if the record is a ray/cell record and the object flag is true if the record is an object/cell record. During the sort distribution operation all the ray flags will be anded together for all records with the same key values. This also happens with the object flags. Therefore any cell that has a record that does not have both the ray flag and the object flag set will be deleted. This cycle of dividing cells and deleting ray/cell and object/cell pairs is repeated until the desired intersection accuracy has been obtained. At this point, object descriptions are distributed to where ray descriptions are, based on the common cells that they intersect. The result is a list of all intersections of rays and objects.

The actual implementation on the MPP uses a 16,384 element-wide stack stored in the staging memory, which is manipulated within the array memory of the MPP. The feasibility of the algorithm was demonstrated by James Hurst on a VAX-11/780 and is being implemented on the MPP.

Additional work needs to done to eliminate unnecessary computation, but this is the first step in the parallelizing of ray tracing on a SIMD architecture.

REFERENCES

1. Dorband, John E., "Sort Computation and Conservative Image Registration," Ph.D. thesis, Pennsylvania State Univ., December 1985.

2. Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp. 15-22.

3. Kajiya, James T., "SIGGRAPH 83 Tutorial on Ray Tracing," *Proc. SIGGRAPH83*, Course 10 Notes, 1983.

4. Kaplan, Michael R., "The Uses of Spatial Coherence in Ray Tracing," *ACM SIGGRAPH '85 Course Notes 11*, July 22-26 1985.

5. Kay, Timothy L. and James T. Kajiya, "Ray Tracing Complex Scenes," *ACM SIGGRAPH '86 Proc.*, Vol. 20, No. 4, Aug. 1986, pp. 269-276.

6. Rubin, Steven M. and Turner Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics* 14(3), July 1980, pp. 110-116.

N87-26557

ANIMATED COMPUTER GRAPHICS MODELS OF SPACE AND EARTH SCIENCES DATA GENERATED VIA THE MASSIVELY PARALLEL PROCESSOR

Lloyd A. Treinish, Michael L. Gough and W. David Wildenhain

National Space Science Data Center
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

The objective of this research is to develop the capability of rapidly producing visual representations of large, complex, multi-dimensional space and earth sciences data sets via the implementation of computer graphics modelling techniques on the Massively Parallel Processor (MPP) by employing techniques recently developed for typically non-scientific applications. Such capabilities can provide a new and valuable tool for the understanding of complex scientific data, and a new application of parallel computing via the MPP. A prototype system with such capabilities has been developed and integrated into the National Space Science Data Center's (NSSDC) Pilot Climate Data System (PCDS) data-independent environment for computer graphics data display to provide easy access to users. While developing these capabilities, several problems had to be solved independently of the actual use of the MPP, all of which are outlined.

Keywords: Computer Graphics, Parallel Processing, Solid Modelling, Animation, Generic Data Representation

BACKGROUND

General

Over the past several years computer graphics has been used to display multi-dimensional (*i.e.*, greater than three) data in support of computer-aided engineering and design, and the film industry via the techniques of solid modelling and animation. These techniques are, in general, very data or application specific, and highly intensive users of computing resources. For example, typical non-scientific applications require almost dedicated use of computers like a Cray X-MP or Cyber 205. To date, such technology has not been commercially available to perform complex, scientific data representation in a generalized and near-real-time fashion, especially at GSFC.

The MPP, which was developed at GSFC, has the potential to process data many times faster than a conventional supercomputer such as the aforementioned Cray or Cyber. If the enormous processing power of the MPP can be applied to the generation of complex computer graphics models then the MPP may have an important impact on the problem of displaying and understanding of complex, multi-dimensional scientific data sets from a variety of disciplines (*e.g.*, space physics, climatology, earth resources, astrophysics) for GSFC researchers.

Many research programs at GSFC generate multi-dimensional data sets from a variety of space and earth science disciplines (*e.g.*, observations of temperature as a function of latitude, longitude, altitude and time). Because of the complexity of such data, it has been quite difficult to gain a complete understanding of their significance. To date, the comprehension of the physics behind the observations has generally been limited to the interpretation of only two- or three-dimensional slices of the data sets of interest because of the lack of appropriate analysis tools.

A data-independent framework for the display of multi-dimensional data sets via computer graphics has been established at GSFC within the NSSDC, as part of the PCDS (Ref. 1). This framework is well-suited for the analysis and display of complex data sets. However, the techniques for performing such data representations beyond simple three-dimensional slices in a useful manner exceed the current capabilities of any computer graphics facility at GSFC. This data-independent framework is supported by the NSSDC Common Data Format (CDF), which is a data abstraction for the source-independent storage and manipulation of data. It provides generic access to structured data and meta-data for applications such as data display (Ref. 2).

Candidate Applications

As a proof of concept, several candidate applications have been selected, from which data sets of interest can be derived, to illustrate the potential of this type of graphical representation for scientific research. The first application is in Middle Atmosphere Electrodynamics (MAE), in which the temporal, spatial and spectral distribution of *bremsstrahlung* x-rays and precipitating electrons during auroral events are studied. Data sets from such observations can fill up to a 6-space model (*e.g.*, x-ray flux as a function of latitude, longitude, altitude, energy and time). Current graphical analysis techniques are limited to subset slices only, for example, spatial distribution of flux at a specific energy level, flux versus energy, flux versus time, etc. It is hoped that through such improved display techniques, a better understanding of energy deposition in the middle atmosphere can be achieved (Ref. 3-4).

A second candidate application is for the generation of simulated mean sea surface maps from satellite altimetry to support oceanography. Traditional display techniques for satellite altimetry cannot show the very subtle variation in the mean sea surface, which relect the composition of the

PRECEDING PAGE BLANK NOT FILMED

lithosphere, and hence, the ocean bathymetry. Solid modelling techniques with appropriate light shading applied to these data have the potential to highlight the detailed structure of the sea surface, in which the light source simulates the action of the active altimeter. It is hoped that through such improved display techniques, a comparison of sea surfaces with ocean bathymetry and gravity fields can be achieved.

Other candidate applications can be derived from the multitude of large, complex data sets managed by the PCDS, whose conventional display techniques are inadequate for some data. In addition, data derived from the LANDSAT Thematic Mapper, could be displayed using solid modelling techniques, when merged with topographic map data.

Representation Methodologies

The NSSDC provides techniques for the conventional representation of arbitrary data, stored in a CDF, in a data-independent fashion, through the PCDS. For one-dimensional data, the display of a histogram would be an appropriate way of illustrating the data. For two-dimensional data, the conventional x-y plot would be appropriate, or a three-dimensional histogram. For three-dimensional data, contour plots, three-dimensional wire-frame or surface diagram, three-dimensional scatter diagram (*i.e.*, x-y-z plot), or a color or gray-scale image would be appropriate. In any of these representation schemes, the use of different coordinate systems or projections may be applied (*e.g.*, cartesian versus polar coordinates, Mercator versus Cylindrical Equidistant map projections, perspective versus orthographic viewing) (Ref. 7).

However, to represent data of greater than three dimensions, as might be available in any of the aforementioned applications, new representation techniques must be developed, in four potential areas. First, for the animation of histories of three-dimensional displays, one would apply any of the conventional three-dimensional representation schemes on a large sequence of the three-dimensional data with respect to some fourth dimension. This concept can be illustrated for an example of global temperature distributions, in which five dimensions of data are available. The first three dimensions are latitude, longitude and altitude, in which this spatial information can be modelled into a three-dimensional wire-frame display. Next, the wire-frame can be filled by a color spectrum, which is mapped to the variation in temperature (*i.e.*, blue implies low temperatures, red implies high temperatures). Finally, a sequence of these solid models of temperature can be brought together by animation to represent the passage of time. Of course, the display can be enhanced by transforming the spatial portion to a map projection of choice, adding a world coastline overlay, graphics arts fonts for annotation, etc.

A second type of data representation, would be in the area of remote sensing scene simulation, in which a complex geometry of instrument movement is used to collect data of interest. This geometry could be combined with the data itself into a complete display that illustrates the nature of the observation geometry at a particular instant or configuration with the appropriate data. A third type of representation would apply to some remotely sensed data that is acquired through active rather than passive sensors, as with radar mapping. Light shading techniques can be used to simulate the characteristics of the active sensor with the resultant observations illustrated as a solid model illuminated by such a light source. A fourth area of data representation techniques, would be for the realistic rendering of remotely sensed data, in which the data of interest is combined with other information to provide sufficient visual cues to help in the understanding of the data. An example of this would be the warping of a classified LANDSAT scene to a topographic map registered over that scene into a three-dimensional solid model.

CONCEPT

This research is motivated out of several factors, one of which, obviously, is the need to provide to the NSSDC community, the ability to display complex, multi-dimensional data sets. However, the NSSDC lacks any significant computational resources, especially for graphics, so an alternative outside of the NSSDC had to be found. Why was the MPP chosen? - essentially because it is there. As mentioned earlier, the MPP has enormous computational potential, which could be applied to graphics. When this work began, the feasibility of developing appropriate parallel algorithms for the MPP was not known. In addition, the configuration of the MPP hardware and software, in principal, allowed it to be the only large-scale computational resource available that is appropriate for its use as a background geometry engine for interactive graphics applications. As it turned out the MPP's hardware and software configuration is not appropriate for such applications, which is discussed below, but no alternatives have yet to become available.

APPROACH

In order to provide an operational capability at the NSSDC to enable users of the data that it supports to display complex data, a systematic approach was developed. The steps in this approach were as follows, but not necessarily pursued in the enumerated order:

- Augment the NSSDC Computer Facility (NCF) with appropriate commerical hardware and software to enable NSSDC software systems to display graphics models.
- Select and prepare candidate data sets within the PCDS via the CDF.
- Devise multi-dimensional representation methodologies appropriate for the candidate applications.
- Develop custom software to link the PCDS to new commerical hardware and software.
- Study feasibility of using the MPP for generation of graphics models.
- Develop modelling algorithms on the MPP.
- Link MPP model generation with data display and management in the PCDS.
- Study and analyze candidate data sets.

This step-wise approach was adopted, so that a useful capability could still be applied to the NSSDC, even if one or more of the steps failed.

ACCOMPLISHMENTS TO DATE

In the implementation of the approach outlined above, the following achievements have been made:

- A development effort to extend conventional two- and three-dimensional graphics within the PCDS data-independent environment to that of true multi-dimensional representations, including animation, has been completed.

- An analysis of typical geometric rendering algorithms has been done in light of the SIMD architecture of the MPP. Some simple graphics algorithms have been developed for the MPP for ray tracing, calculation of surface normals, etc., and shown to be adequate for basic geometric rendering applications. However, for an operational environment, a surface fitting algorithm has been developed, in which a large sequence of arbitrary triples of data (*i.e.*, [x, y, z]) are sent to the MPP, and an empirical functional relationship (*i.e.*, z[x, y]) is established.

- An extensive survey of the advanced computer graphics workstation market has been conducted, which led to the competitive procurement of a Megatek Merlin 9200, which has been integrated into the NSSDC Computer Facility (NCF) via its local area network and hence, SESNET, for direct communications to the MPP. Figure 1 illustrates the configuration of this hardware, and the communications links that support the the transfer of data to and from the MPP and the Merlin 9200.

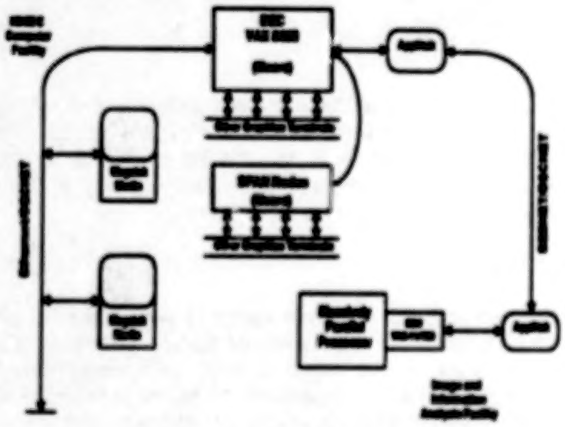


Figure 1. Hardware Configuration for MPP Graphics at the NSSDC

- Several generic data representation methodologies have been developed for the display of complex, geophysical data sets, which have been implemented via the MPP and the NCF.

- The software on the MPP and the NCF have been integrated with the workstation environment to permit a potential user of these advanced graphics techniques to easily get at and work with data of interest. Basically this has involved the tying of these display techniques on the MPP with the PCDS in a transparent fashion, and the establishment of a user-friendly environment for these tools. Figure 2 illustrates the configuration of this software in the NCF environment, in which several NSSDC software systems access a generic graphics system for data representation, and utilize the Merlin 9200 as well as other graphics hardware for actual displays.

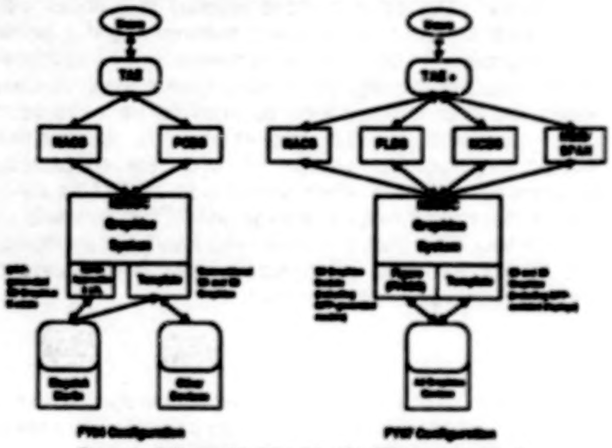


Figure 2. Software Configuration for MPP Graphics at the NSSDC

- Some candidate data sets have been identified from which complex, graphics models have been generated, in the areas of solar-terrestrial physics, climatology, and oceanography.

MPP ALGORITHMS

Obviously, the MPP has enormous computational power, but its SIMD architecture is really not the optimal one for graphics modelling. In fact, efforts to utilize a parallel architecture for computer graphics have only begun fairly recently (Ref. 8-9). Therefore, an investigation and evaluation of different graphical techniques was begun for the MPP. This effort began with an examination of traditional graphics rendering algorithms (*e.g.*, ray tracing, depth sorting, area subdivision, z-buffering, etc.). Some experiments were done with brute forcing ray tracing, using the array as a z-buffer for hidden line and surface removal to prove the feasibility of using the MPP for graphical rendering. However, given the limitations implied by the current hardware and software environment to support the operational scenario, which is outlined in a subsequent section, it was decided to utilize the MPP for only partial rendering of the data of interest. In this situation the MPP is used to calculate surface meshes from arbitrary data, from which a final rendered display is generated at the NCF.

Within this revised environment, the MPP basically performs two functions. First, it is used to construct a three-dimensional surface on a uniformly spaced grid of specified resolution from a non-uniformly distributed set of arbitrary input data points. Second, it computes vectors, which are normal to the three-dimensional surface at each of the grid locations. The SIMD architecture of the MPP is used to perform these calculations simultaneously for each grid point. Several different gridding techniques have been implemented and tested with a random distribution of data points, which are outlined below. Two of the methods, which compute a local surface function for each grid point (methods 6. and 7. below), were first implemented on the NCF DEC VAX 8650 for testing. They were found to be inadequate because they resulted in excessive surface undulations. The other methods resulted in surfaces with much better behavior. In particular, methods 4. and 5. below seem to provide the best surface representation. An additional method was also investigated, in which bicubic Bezier patches were used to constructed local surfaces for each grid point based upon the closest 16 data points (Ref. 10). However, this method required the solution of two very complicated simultaneous equations, which proved to be beyond the ability of the symbolic mathematics package, MACSYMA, available on the NCF VAX 8650 (Ref. 11). There was insufficient manpower and funding resources to attempt to solve these equations using an interative numerical approach.

Step One

The construction of a gridded three-dimensional surface from a non-uniform distribution of arbitrary data points was attempted via seven different techniques enumerated below:

1. Nearest Point Algorithm

The Euclidean distance between each incoming data point (represented by an arbitrary 3-tuple [x,y,z]) of the non-uniform distribution and each grid point is computed to find the closest data point to each grid point of the uniform grid. Each grid point is then merely assigned this z-value.

2. Average of All Data Points Weighted by Inverse Distance

The Euclidean distance between each incoming data point and each grid point is computed. Each grid point is assigned a z-value based upon a weighted average of all of the incoming data points, with the weights being assigned on the basis of the distance between the data point and the grid point. The closer points are given a higher weight than the farther points since the inverse of the distance is used as the weight.

3. Average of All Data Points Weighted by Inverse Squared Distance

This method is identical to method 2. except the weight is based upon the inverse square of the distance between the data point and the grid point.

4. Weighted Average of the 5 Closest Data Points to each Grid Point

The Euclidean distance between each incoming data point and each grid point is computed to determine the 5 closest data points to each grid point. A z-value is then assigned to each grid point based upon a weighted average of these 5 closest points, with weighting being done with the inverse of the distance.

5. Weighted Average of the 3 Closest Data Points to each Grid Point

This method is identical to method 4. except that 3 points are used rather than 5.

6. Construction of a Hyperbolic Surface for each Grid Point

The Euclidean distance between each incoming data point and each grid point is computed to determine the 6 closest data points to each grid point. A 6 x 6 matrix, A, is then constructed for each grid point with the following basis functions:

$$\begin{array}{lll} 1 & x & y \\ xy & x^2 & y^2 \end{array}$$

Using the z-values of the 6 closest points as the right-hand side, b, the following can be established:

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}, \qquad (1)$$

which can be solved for x, to yield the coefficients of a local hyperbolic function. Each function is then sampled at the grid location to generate a z-value for that point.

7. Construction of a Simple Surface for each Grid Point

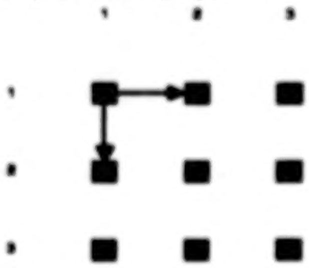
The Euclidean distance between each incoming data point and each grid point is computed to determine the 4 closest data points to each grid point. A 4 x 4 matrix, A, is then constructed for each grid point with the following basis functions:

$$\begin{array}{lll} 1 & x & y \\ x & xy & \end{array}$$

Using the z-values of the 4 closest points as the right-hand side, b, then equation (1) for a 4 x 4 system can be utilized, which can be solved for x, to yield the coefficients of a local surface function. Each function is then sampled at the grid location to generate a z-value for that point.

Step Two

The construction of a normal vector at each grid point of the uniform, gridded three-dimensional surface calculated in Step One is done in two steps. First, two non-colinear line segments from each grid point on the southward and eastward directions are constructed, which is illustrated in the schematic below, in which the two line segments are constructed from the extreme northwest point in a sample grid:



Let the label, *A*, designate the line segment or equivalent direction vector, extending from the grid point (1,1) to grid point (2,1), in this example, where the notation (row, column) is utilized. This direction vector is formed by merely subtracting the [x,y,z] components of point (1,1) from the [x,y,z] components of point (2,1). Let the label, *B*, designate the line segment or equivalent direction vector extending from grid point (1,1) to grid point (1,2). This second direction vector is formed by subtracting the [x,y,z] components of point (1,1) from the [x,y,z] components of point (1,2).

Second, take the cross-product of the two line segments from each grid point. In the above illustration, this would imply *A* x *B*, with the resultant vector being perpendicular to the plane defined by the two line segments, *A* and *B*. This resultant vector is the normal vector to that particular grid point, and is used by the NCF hardware and software to render a final display. This procedure is carried out simultaneously for each of the grid points of the surface mesh, with the exception of the last column and the bottom row of the grid, which require northward and westward constructions of the line segments.

OPERATIONAL SCENARIO

The aforementioned tools have now been integrated at the NCF in the graphics workstation environment to permit a user to view such complex models of data via the same, simple mechanism that a user can view data of interest in the PCDS environment in a conventional two or three-dimensional form. This system represents the first *operational* use of the MPP, in the context of an extant user-friendly software system. Figure 3 is a schematic of the information and data flow that supports this operational graphics capability, in the context of the hardware and software configuration shown in Figures 1 and 2, respectively. In this system a user creates or generates data of interest in terms of CDF for the PCDS. A user can then create conventional representations of two or three-dimensional slices of the data. However, a user can also generate a solid model representation. When that happens, the data of interest from the CDF, and the model request are shipped over to the MPP via SESNET, where a batch job is submitted under interactive control at the NCF. The MPP software generates the surface rendering, and ships it back to the NCF for final display and rendering on the Merlin 9200. Animation is achieved via the display of sequences of such models.

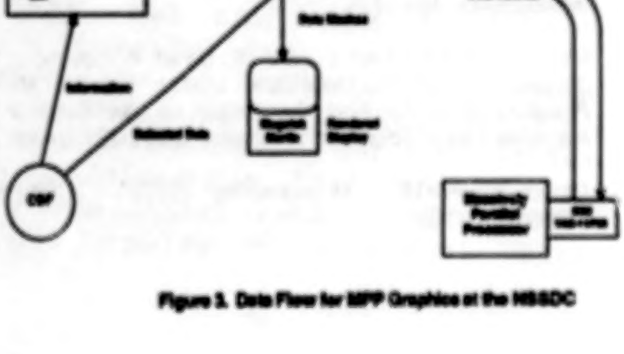


Figure 3. Data Flow for MPP Graphics at the NSSDC

At this point any member of the MPP Working Group is cordially invited to view this capability in action, and to evaluate its utility to supporting their research.

MEGATEK MERLIN 9200

The Megatek Merlin 9200 is an advanced solid-modelling graphics terminal, that is oriented around a hierarchical three-dimensional architecture for graphics, and a distributed processing concept. This terminal supports a 3072 x 2304 display resolution with 12-bit planes of double buffered display memory. It is capable of processing full three-dimensional transformations on 50,000 vectors per second or 3800 shaded polygons per second. Within its three-dimensional graphics capabilities it can support a data base of up to 4 MB with a dedicated processor. Its graphics processor also supports light shading via flat, Phong and Gouraud algorithms. It has a separate local task processor, which can be programmed independently from the host, supported with 1 MB of memory. The Merlin relies on the NCF VAX 8650 as a host computer, which communicate via a DECNET/Ethernet connection for graphics, and conventional asynchronous communications for alphanumerics. The Merlin also supports multiple input devices, including three valuators, a keyboard, joystick, and data tablet. The surfaces calculated on the MPP are downloaded to the Merlin graphics database, and then rendered by the Merlin itself. In this sense, the VAX 8650 acts as a user interface, and communications handler between the user, the MPP VAX-11/780 and the Merlin (Ref. 12).

RESULTS

This research has yielded a new operational capability to display complex data sets in a novel fashion. At this time, the tools are just beginning to be used to analyze space and earth sciences data of interest. Color Plate V is an example of the type of graphical product that the system can generate. It illustrates the spatial and spectral distribution of gridded energy fluxes calculated from remotely sensed x-ray counts observed from a sounding rocket instrument during an auroral event (Ref. 4).

Problems

While doing this work several unforseen problems arose, which has delayed the completion of all of the proposed activities, and has prevented the completion of the actual analysis of the candidate data sets via their geometry, which was originally planned for the current fiscal year. There were two major problem areas that were encountered. The first problem is related to the MPP itself. The MPP software environment lacks sufficient flexibility and ease of use to permit the development of an operational MPP-based graphics system (Ref. 13). The current MPP environment is only suited for the clumsy development of *one-shot* code, and not for the development of real applications systems. Therefore, a conceptual framework for such a proper environment was developed and implemented to permit simpler access to the MPP via a software *toolbox* as a collection of procedural abstractions, which includes a virtual memory manager, and transparent task-to-task communications across SESNET or the Space Physics

Analysis Network (SPAN, (Ref. 14). This MPP task funded and supervised the development of this toolbox. This software can be made available to MPP Working Group for use in any applications software to provide for the first time reasonable access to the MPP's enormous computational power. The details of this software development effort and the resultant package is described in a companion paper to this one, A **Generic Applications Subroutine Library for the MPP** by Michael L. Gough and W. David Wildenhain (Ref. 15). Of course, this toolbox permitted the completion of the graphics applications.

The second major problem area was in procurement. The acquisition process for the advanced computer graphics workstation necessary to support this work required over a year. This delayed the development of system-specific software, and learning how to employ such a state-of-the-art hardware/software system.

PLANNED FUTURE WORK:

It is expected that this work can continue in several different areas if funding is made available. The efforts described herein, were funded primarily by the NASA/GSFC Director's Discretionary Fund for FY85 and FY86. However, there is no currently available funding to continue this work, even though it is still part of an approved MPP Working Group project.

Of course, the actual analysis of the candidate data sets should be supported as part of any continued research effort. However, much interest has been generated in these graphical modelling techniques for a wide variety of space and earth science data sets, especially those that are managed by the NSSDC. Hence, one would like to apply these display techniques to a number of data sets that the NSSDC manages in a variety of disciplines. Therefore, the tools developed to date will be *turned over* to the NSSDC for use in its operational PCDS-supported research, for example.

In addition, further efforts in the exploration of computational power of the MPP, in the context of the MPP Working Group, should be pursued. In particular, the development of other data representation schemes coupled with the development of other rendering algorithms offers a number of challenging ideas. For example, revisiting the idea of implementing surface fitting via bicubic Bezier patches would be of great interest.

Although the implementation of these tools has been established for the Megatek Merlin 9200 hardware, it should be recast in terms of the proposed ANSI standard for PHIGS, the Programmer's Hierarchical Interactive Graphics Standard, to provide computer as well as device portability, with high-level three-dimensional functionality. This will permit the future use of these applications in a wide variety of environments, including other high-end graphics systems for the display and manipulation of such models (Ref. 16).

REFERENCES

1. Treinish, Lloyd A. *A General Scientific Information System for the Study of Climate-Related Data*. NASA Technical Memorandum, September 1984.

2. Treinish, Lloyd A., Michael L. Gough, C. Edwin Wilson. *A Software Package for the Data-Independent Management of Multi-Dimensional Data*, NSSDC, September 1986.

3. Goldberg, R. A., C. H. Jackman, J. R. Barcus and F. Soraas. "Nighttime Auroral Energy Deposition in the Middle Atmosphere." *JGR*, *89*, pp. 5581-5598.

4. Goldberg, R. A., J. R. Barcus, L. A. Treinish and R. R. Vondrak. "Mapping of Auroral X-Rays from Rocket Overflights." *JGR*, *87*, pp. 2509-2524.

5. Koblinsky, C. J., J. G. Marsh, C. Nethery and V. Zlotnicki. *Development of a Graphical Analysis System for the Study of the Earth and the Oceans*. A Proposal Submitted to the Director of NASA/GSFC, 1984.

6. Marsh, J. G., C. J. Koblinsky, A. Brenner, B. Beckley and T. Martin. "Global Mean Sea Surface Computations Based upon a Combination of SEASAT and GOES-3 Satellite Altimeter Data." *EOS*, *65*, 45, p. 857.

7. Reph, Mary G., Lloyd A. Treinish, Carey E. Noll, Thomas D. Hunt and Shue-wen Chen. *Pilot Climate Data System User's Guide*. NASA Technical Memorandum, January 1986.

8. Dippe, Mark and John Swenson. "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis." *Computer Graphics*, *18*, 3, ACM Siggraph '84 Proceedings, pp. 149-158.

9. Levinthal, Adam and Thomas Porter. "Chap - A SIMD Graphics Processor." *Computer Graphics*, *18*, 3, ACM Siggraph '84 Proceedings, pp. 77-82.

10. Foley, J. D. and A. van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982, pp. 505-538.

11. Symbolics, Inc. *VAX UNIX MACSYMA Reference Manual*. Version 11, November 1985.

12. Megatek Corporation. *Merlin 9200 User's Guide*. Version 1.0, February 1985.

13. Fischer, Jim. *MPP User's Guide*. NASA/GSFC, January 1986.

14. Green, James L. and David J. Peters. *Introduction to the Space Physics Analysis Network (SPAN)*. NASA Technical Memorandum, April 1985.

15. Gough, Michael L. and W. David Wildenhain, "A Generic Applications Subroutine Library for the MPP." *Proceedings of the First Symposium on the Frontiers of Massively Parallel Scientific Computation*, NASA/GSFC, 1986.

16. TEMPLATE. *Understanding PHIGS*. Megatek Corporation, 1985.

N87-26558

REAL TIME ANIMATION OF SPACE PLASMA PHENOMENA

by

K. F. Jordan and E. W. Greenstadt

Space Sciences Dept., TRW, Redondo Beach CA 90278

ABSTRACT

In pursuit of real-time animation of computer simulated space plasma phenomena, we have rewritten code for the MPP. The program creates a dynamic representation of the global bowshock which is based on actual spacecraft data and designed for three-dimensional graphic output. This output consists of time slice sequences which make up the frames of the animation. With the MPP, 16384, 512 or 4 frames can be calculated simultaneously depending upon which characteristic is being computed. The run time has been greatly reduced which promotes the rapid sequence of images and makes real-time animation a foreseeable goal. The addition of more complex phenomenology in the constructed computer images is now possible and work proceeds to generate these images.

INTRODUCTION

With several spacecraft making measurements in space, the ability to explore global phenomena is greatly increased. However, analysis and integration of the massive amounts of data require new techniques for gaining insight into the optically invisible, space plasma phenomena. Three dimensional real time animation will be very powerful tool and will allow scientists to correlate data beyond merely comparing static plots of multiple variables looking for similarities or dissimilarities. Traditional computing capability cannot deliver the rapid display of sequential which simulate "views" of the earth's plasma layers as they change with solar wind conditions. In our initial effort, data from the ISEE-1 and ISEE-3 spacecraft are used to create a dynamic three-dimensional model of the earth's bowshock. Real time animation (which we define as a frame every images second or less) will allow the scientist to interactively manipulate and verify his model. This interaction will consist of accessing any desired data set, changing model equations and constants, and inspecting selected features. At present, we think the MPP very promising for achieving our objectives and beyond.

PROBLEM

Animation is notoriously demanding on memory and machine speed to acheive realistic motion directly from the computer itself. For our previous production, we used a VAX 750 and Evans and Sutherland Multi Picture System. Typically, a 350 frame film took 3 1/2 hours to run. The time factor forced us to compute the entire sequence of data and store the bowshock variables onto magnetic tape. Then, the images were rendered based on the data on tape and filmed frame by frame. The chemical film processing took at least a week and the film to-video conversion took at least two weeks. Turnaround time was further increased due to the raw data being on magnetic tapes that were specially made and shipped to us.

N87-26558

The problem is to eliminate the multistage process, the unacceptable executions times, and the uncontrollable multimedia interfaces between the data input and the animated output scene. We would like to eliminate chemical stages altogether.

APPROACH AND RESULTS

Communications. Since our offices are on the west coast, our interactions with the MPP system must take place through one or more communication paths. We have relied heavily on the Space Physics Analysis Network (SPAN) for transferring code and testing results. Telephone conversations with our assigned MPP advisors and a 7 day trip to the MPP facility also allowed work to progress. We favor SPAN because, aside from being charter organizers and supporters of the system, we routinely employ the network to obtain the satellite data used as input to the graphic models we want to animate.

Image Computation. Our initial effort with MPP has been to reproduce the animation sequence already completed with the former system. A frame of the animation represents the Earth's bowshock which is depicted as discrete rings. The number of rings are determined by the data sampling rate and the hyperbolic model. The rings are divided into quasi-pendicular (green) and quasi-parallel (blue) sections based on local magnetic field and the normal to the shock in the B-X-plane. The "predicted" bowshock structure is based on and verified by spacecraft data from ISEE-3 (density, solar wind parameters, magnetic field vectors) and ISEE-1 (shock structural observations). Each frame of the animation is independent and requires reading and solving its own set of data values. There are 32 to 4096 values to be calculated for each variable per frame. Due to computational time restrictions, the model is very simplified. Many assumptions are made about solar wind delivery angle and delay times. We were confined by low data sampling rates and low resolution inputs. Both the model and its assumptions could be tested by recycling our animation sequence through changes in parameters and algorithms.

In the MPP system, we still separate the image computation from the display process, but whole frames of 4096 points can now be calculated together instead of only one point at a time. Figure 1 shows how the variables can be mapped to the MPP parallel arrays. The variables that are related to the 32 rings per frame are calculated 512 frames at a time, others that are related to the points can be calculated 4 frames at a time. The MPP only takes 10 minutes to compute 350 frames (A vast improvement from the VAX). The majority of the time is spent storing data to and from disk.

Image Production. With the number crunching aspect solved, the remaining problem is the graphics. There are no facilities on the MPP to create the picture. Through the SPAN network, the data files from the MPP are transferred to other computers (such as NSSDC) to be rendered on local terminal. The data files require 512 blocks plus 384 blocks for each group of four images. Since the graphics through the network consumes much storage space and takes too much time for real time animation, it is currently part of the test process only.

CONCLUSIONS

The capabilities of the MPP are bringing us closer to the goal of interactive animation. The MPP and its access through SPAN allow increases in both available data sets and model complexity. Any data set can be requested from UCLA, JPL, or other data nodes through the network rather than by magetic tape. With minutes rather than hours of computation time, the model equations can be adjusted (and will be in the future). With the dramatic increase in speed realized by the MPP, the scientist's interaction with the data through real-time animation becomes more than just a pipe-dream.

FUTURE

Graphic Improvements. We plan to include raster representations. The graphics will continue to be done by passing the data files through the network but we will use a local computer to allow real time performance. An affordable means of recording from graphic monitor to video tape is desirable to easily preserve the results for presentation to the scientific community.

Model Improvements. As soon as regular display of MPP-produced scenes is implemented at TRW with acceptable turnaround times, we shall begin altering the parameters, assumptions, and algorithms of our existing model to find better conformity between predicted and measured phenomena.

Model Enhancements and Enlargements. We will be adding complexity to our models to allow more than two shock structural types, several foreshock surfaces, magnetopause depiction, and magnetosheath dynamics.

ACKNOWLEDGMENTS

We would like to recognize the assistance and expertise of the NASA MPP consultants. Ed Seiler and Jim Hurst were especially helpful as our assigned support. Mr. Seiler's working hours were a special boon to those of us working on the West Coast. We would also like to thank Lloyd Treinish for his help with NSSDC and Template and also to SPAN for allowing us to do any work at all on the MPP.

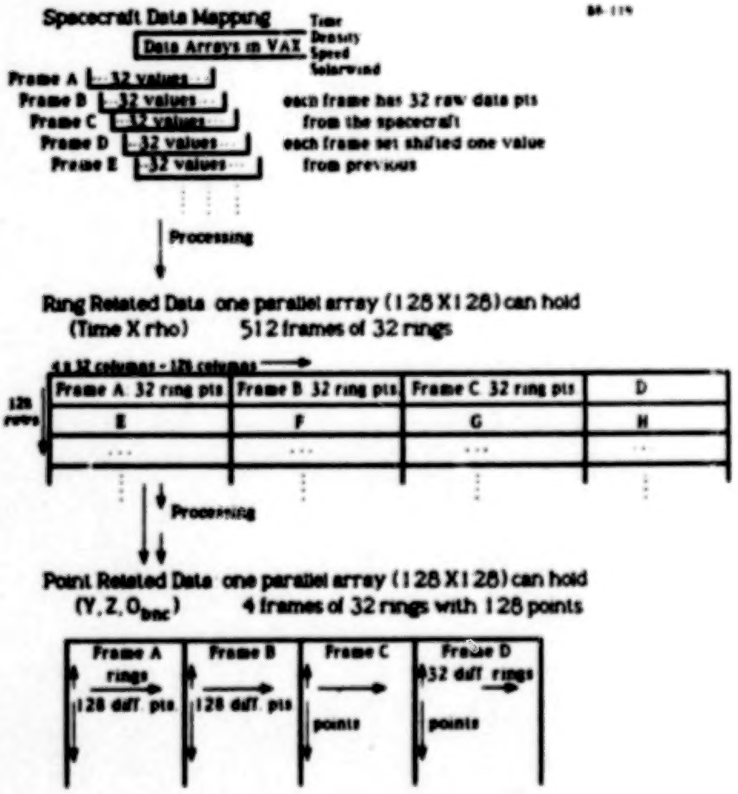

Spacecraft Data Mapping
Data Arrays in VAX
Time
Density
Speed
Solarwind
Frame A ...32 values...
Frame B ...32 values...
Frame C ...32 values...
Frame D ...32 values...
Frame E ...32 values...
each frame has 32 raw data pts from the spacecraft
each frame set shifted one value from previous
Processing
Ring Related Data one parallel array (128 X 128) can hold
(Time X rho) 512 frames of 32 rings
4 x 32 columns = 128 columns
128 rows
Frame A 32 ring pts
Frame B 32 ring pts
Frame C 32 ring pts
D
E
F
G
H
Processing
Point Related Data one parallel array (128 X 128) can hold
(Y, Z, O_bnc)
4 frames of 32 rings with 128 points
Frame A
rings
128 diff. pts
Frame B
128 diff. pts
Frame C
points
Frame D
32 diff. rings
points

Figure 1 Mapping of Parallel Arrays

N87-26559

Algorithmic Commonalities in the Parallel Environment

Michael A. McAnulty and Michael S. Wainer
Department of Computer and Information Science
University of Alabama in Birmingham
Birmingham AL 35294

ABSTRACT

The ultimate aim of this project is to analyze procedures from substantially different application areas to discover what is either common or peculiar in the process of conversion to the MPP. Three areas were identified - molecular dynamic simulation, production systems (rule systems), and various graphics and vision algorithms. To date, only selected graphics procedures have been investigated. They are the most readily available, and produce the most visible results. These include simple polygon patch rendering, raycasting against a constructive solid geometric model, and stochastic or fractal based textured surface algorithms. Only the simplest of conversion strategies, mapping a major loop to the array, has been investigated so far. It is not entirely satisfactory.

Key Words:Graphics, Stochastic Surface, Constructive Solid Geometry, Conversion Strategy, Fractal Surfaces

INTRODUCTION

Intuitively, certain procedures appear to map easily to an array machine, and others do not. The latter class includes those with long stretches of inherently single-thread code, for which even incredible speed-ups in their parallel part will produce little net improvement, and those whose internal communication structure is irregular, such as highly general neural nets. The three classes to be discussed were chosen not for their impossibility, but rather for their *prima facie* differences, as well as for their presumed availability during this work.

Molecular dynamics simulates the motion of complex macromolecules, which can be shown to exhibit a quite interesting ensemble of configurations. The simulation relies upon the locality of effects, so that each atom interacts only with a few neighbors, and accuracy and stability are sought by using a sufficiently small time step. The locality of interaction and the uniformity of the calculations for each atom appear to make this mappable to an array machine, albeit with some significant programming work. We intended to use some existing code which has been vectorized, but this is now being marketed, and we are still discussing the possibility of using it without violating its proprietary nature.

Rule systems invoke a branching chain of inferences using a domain rule data base suitably encoded. A rule is activated by determining that certain facts in a fact data base satisfy its left hand side, and it executes by making modifications to the fact data base. Some variant of this underlies most expert systems currently available. As it happens, the expansion of an inference is often a highly branched tree process. The necessary depth of a search is the discouraging component of the process, but this is often outweighed by a considerable breadth as well. The major bottleneck in production systems appears to be, in fact, not the depth of the process, but its breadth which is necessarily invoked in passing left hand sides against the fact data base to detect satisfied rules, a process termed matching. Noting that, with the broadcast facility, an array machine may function reasonably well as a large associative memory, it appears possible that rule systems could be significantly optimized in an array machine. Unfortunately we have lost the investigator who was to have pursued this line, using an existing production system, and it has not been followed up yet.

The final area, graphics and vision algorithms, is represented only in the graphics area. We may plead, citing a fairly common observation, that in fact vision and graphics appear to be the same process run in two different directions between an image and its description (Ref. 7), but in fact they are nowhere near that in practice. A short section will address vision issues. The bulk of the following represents graphics issues that have emerged during our involvement with the MPP.

POLYGON PATCHES

The description of a scene as a collection of simple patches that together describe surfaces of solids is perhaps among the oldest of solid models, and a discussion of its aspects may be found in several

PRECEDING PAGE BLANK NOT FILMED

comprehensive references (e.g., Ref. 1). There is a considerable leap between models with simple planar patches and those with simple curved patches, this discussion treats only planar patches. The process of mapping a patch to an array of display elements (pixels), which is termed *scan-conversion*, is relatively straightforward, if tiresome, for planar patches, and not nearly so for nonplanar patches.

To render (that is, display) a polygonal patch requires projecting it to the two-dimensional display surface, a simple scaling and shifting operation, and then determining which pixels it in fact projects to. This last operation, the actual scan-conversion, will usually consist of counting from one boundary to the next in pixel-sized increments. It is roughly as time consuming as the number of pixels actually covered, although there is also some overhead involved in setting up the control parameters of the loops involved. Polygons are as easily represented as a combination of half-spaces, simple linear constraints, as they are by vertices, and one form may be generated from the other with comparable facility. A quite straighforward array process for scan conversion may be described, which consists of each array element testing its location against the list of half spaces, and 'lighting itself up' if it passes all the constraints. There is an added advantage to working on the array, in that each processor may serve as a 'z-buffer', keeping track of the depth which its current pixel represents (if indeed it has been filled). This can be simply calculated from the 3-space plane of the polygon, and is useful when patches overlap. If a new patch does project on a processor at a nearer depth, it simply wipes out the previously stored value.

The simple (but effective) method of passing linear constraints to the array, one polygon at a time, is in fact no less onerous than it would be to pass them to some other firmware scan-converter, the usual process in the sequential world. The process is advantageous if rather large polygons are involved, the scan-generation process becomes a single 'multi-step', but one is reminded that during this step most of the array elements are not doing 'useful' work.

An alternative approach, to somehow load the array with patch definitions and let each scan generate in parallel, confers both advantages and considerably more work. The advantage is that an animated sequence, where the collection of patches moves in time, does not require a sequential scan of the patch data base for each new frame. The challenge is that scan conversion is no longer as direct, a process on the array may not be located in the pixel it is eventually supposed to illuminate, and thus the values the process generates must then be routed to the correct pixel. We have concentrated on processes that map array elements directly to pixels, and the costs of a general routing process remain to be more carefully considered. It should be noted that the hidden surface problem is still nicely addressed, as destination pixels may still function as z-buffers.

The simple half-space approach corresponds to a general methodology, unwrapping a controlling loop and spreading it over a processor array. In the planar patch model, the innermost loops that direct scan generation are the ones unwrapped. The outer loop, which counts through the model data base exactly once (since occlusion is handled by the z-buffer), is totally determined. The inner loop in fact counts over pixels, but in a piecemeal and unpredictably repetitive fashion. An alternative approach is to count over pixels, hitting each exactly once, and for each pixel traverse the model data base to see which surface, if any, projects to that pixel. This is the control structure of the next rendering method to be discussed.

RAYCASTING SOLID MODELS

For complicated but man-made objects the constructive solid geometry (CSG) models are unsurpassed for direct representation and conciseness. They are particularly well-adapted to rendering by ray-tracing, which is still the best method with which to exercise full control over the lighting model, including shadowing, matte and specular reflection, and refraction. (see, for example, Ref. 10) In the single thread implementation, the major loop counts through all pixels, constructs a ray through each pixel, and determines its intersection with the model. The outer loop, once per pixel, can be mapped onto the processor array quite easily, so that each element holds a different ray, and the model may be broadcast one node at a time. The model need be traversed only once (but see further), which is of some modest advantage for CSG because models tend to comprise components numbering from scores to hundreds. However, for other models such as oct-trees, where model traversal is more expensive, this advantage increases.

The operation at each pixel, intersecting it with the solid model, is directed by traversing the model, formed as a tree, in preorder. Leaf nodes represent one of a small set of primitives (cube, sphere, cone, torus), interior nodes represent set combinations of their subtrees, which may be intersection, union, or subtraction, and these are always binary. Each visit to a leaf produces a list of intersections, always in in-out pairs. The various intersections are represented as a parameter value along the ray, with larger values being further from the viewer. The grazing phenomenon, where a ray touches an object once, can always be forced out at this level. Each visit to an internal node produces two such lists which must then be combined into a single list according to the operation represented at the node.

It would be sufficient to retain only the nearest in-out pair if it weren't for the quite powerful set subtraction operation- no matter how many hits one saves, they may all be subtracted further up the tree, and it may turn out that one needed just the ones thrown away. It is possible to force subtraction to be done at low levels in the model tree, which will alleviate this problem, although the general control structure remains unchanged. At any interior node in a tree, several lists of intersections from left subtrees at higher levels may be in effect, waiting to be combined with right subtrees, and each list may be zero or more intersections. There will, if empty lists are explicitly treated, be exactly the same number of stacked lists for each ray (processor), so that a combine operation which mixes a current right branch intersection list with the topmost left intersection list may be done in parallel, with processors that exhaust their left list simply shutting off for the duration of the combine. Because there should be no limit on tree depth, there is thus no limit on the stack size even if we restrict a single intersection list to two members, thus it appears that we shall require the staging memory to stack the lists. More onerous, since a parallel reference must be to the same address in PE memory (and by extension in the stager as well), it appears that stack operations should physically relocate the entire current stack. The solution to this problem is still under consideration, and may perhaps be addressed by restricting single lists to pairs, which may have real values or dummies that indicate no intersection.

The stacking problem relates only to the traversal of the model. When this has been accomplished, only the foremost intersection matters. Any further work to be done on that pixel consists of casting further rays from the intersection point. We may need to cast shadow feelers to light sources, reflected rays from specular surfaces, or refracted rays at non-opaque surfaces. It becomes necessary, then, to include with all intersections in the stack the physical properties of the object being intersected, which include color, transmittance, specularity, and the surface normal of the object at the intersection point. More important, we require parallel management of these further raycasts.

Shadow feelers, for simple shadowing, are straightforward since every intersection must cast to the same light sources, combining the results of the cast with its matte reflectance and color. At this point, non-specular and non-transmitting intersections may turn themselves off. Those intersections which are specularly reflective to any degree may cast only once, unless this second ray also encounters a highly reflective surface. For many applications we might restrict the number of times this might occur. A similar consideration applies to transmitting surfaces, which could be treated as a pair of in-out rays with a final modulated intersection.

The extra contributions to a pixel's value from refracted or reflected objects need to be combined in weighted fashion to calculate the value. This applies as well to further recursive extensions of such feelers. At each extension, the net contribution of a particular feeler to the ultimate value of the original pixel, its weight, grows less and less. Thus, we may generate as many rays as we need, attach to each its weight, and continue generating feelers until no new feeler has a weight above some threshold. Each new feeler results in the stacking of weighted values, which will generally have zero weights for dull surfaces, and combination may be a straightforward stack operation. However, this results in a doubled data requirement, and as many extra model traversals as extra rays are generated. The anticipated speed-up of 16,000 will not be attained, since only a small minority of the rays, in general, require further casting, so the majority of rays are idle (processors doing no useful work) while reflectors and transmitters are followed, each stage of which requires a further sequential traversal of the model.

One might, in the case of a really fancy lighting model offload certain rays to other processors which are no longer propogating, but the nature of routing such rays appears to be infeasible, and there is in fact little parallelism in the feelers, one must generally follow another.

Interim Discussion

The half-space polygon rendering model has been implemented and run, in highly primitive fashion, on the MPP, largely as an exercise. The CSG raycaster is still an incomplete paper exercise. The only conversion methodology applied in either case is the quite straightforward unwrapping of major control loops. The major challenge appears to be manipulation of model elements entirely within the array. This was alluded to in the discussion of patch models, and is necessary for anything like real-time manipulation, in three dimensions, of the model. Before an actual implementation of the ray-caster is attempted, further study of its internalization will be done.

While "late vision" algorithms have some relation to graphic modelling, a major challenge at this stage is the matching of elements of the model with the image, an interesting problem we have yet to deal with. Early vision procedures are of lively current interest, and tend to be highly parallel and local computations involved in regularization and related relaxation procedures. (Ref. 9) These appear, intuitively, to be "easy" procedures to map to a matrix machine. However, the graphics procedures we have dealt with also appear easy in much the same sense. The initial methodology has been to do those easy tasks and see what stumbling blocks in fact appear. For both patches and CSG neither "easy" mapping has been entirely satisfactory. Further, there is a strong possibility that many algorithms may be too numerical in nature and realization, numerical computation having been our major medium in the past. We should investigate more efficient uses of connected bit-serial processors which minimize arithmetic.

The final and dominant segment of this project (Wainer) maps a different process to an array in some detail, and has been fully implemented. The control structure is again recursive, but unlike the CSG traversal the recursion itself, rather than a "major loop", is mapped directly to the array. Further, some use is finally made of the communication structure of the array.

STOCHASTIC INTERPOLATION OF RECTANGULAR PATCHES

Stochastic interpolation of rectangular patches as described in (Refs. 2,3,8) is useful in generating dense patch data from a small number of input values. In computer graphics the method is used to create surfaces, textures and sky among other things (Refs. 2-4,8). The values generated are pseudorandom and approximate a fractal distribution. The characteristics of the surface are controlled by the h and scaling parameters. Values of h near 1 give a smoother surface than those near 0. The scaling parameter is used to adjust the magnitude of the stochastic contribution to the interpolation function. For more about fractional brownian motion and its applications see references 3 and 8.

The process begins with a parent patch whose corner data values are given. These values may be interpreted as altitudes or colors or some other attributes which are to be interpolated across the patch. The parent patch is repeatedly subdivided until the child patches are of the desired resolution (ie. pixel sized). The interpolated values are formed from the deterministic average of neighboring data values and a stochastic component which is determined by the parameters and pseudorandom numbers.

A graphical depiction of the method is shown in Figure 1. The four corners of the parent are used to calculate the center point of the child. Side points along an edge of the original parent use only the data derived from the two original points which define that edge. Side points in the interior of the original patch use the four values from their vertical and horizontal neighbors. By stipulating that points that lie along the edges of the original patch depend only on the vertices that define those edges, continuity between adjacent input patches can be guaranteed.

The stochastically interpolated values are formed from both a deterministic and a stochastic component.

$$I = I_{det} + I_{stoch} \tag{1}$$

I_{det} is the average of two or four neighboring vertices as shown in Figure 1. The same neighbors which determine I_{det} are used to create a seed value for I_{stoch}. A pseudorandom number is selected using this seed and is conditioned by the generating parameters h and scale. This method of forming the seeds for pseudorandom number generation assures continuity between adjacent patches as mentioned above. To prevent the selection of pseudorandom numbers solely on the basis of the original parent patch's corner data values, seed values are also provided with each original data value. Points whose data values were given at the start of the procedure supply seed values for I_{stoch} of their dependents.

Complexity of the Process

It is easy to see that the number of patches grows exponentially in powers of four. Even though patches share some of the same vertices, we cannot better the overall complexity of the process by more than a constant factor. (Consider that each patch in the interior has 1/4 + 1/4 + 1/4 = 3/4 new points to calculate: this is true in every generation. Patches on the edge must generate 1/2 + 1/2 + 1/4 = 5/4 new points.) Thus whatever sequential algorithm may be used, its time complexity will still be at least $O(4^L)$ where L is the number of subdivision levels. Furthermore even though the algorithm appears inherently parallel, the data dependencies of children on their parents limits the best we can do to an algorithm which has a time complexity proportional to the number of subdivisions or equivalently to the log base 4 of the number of final patches. The data dependencies make matters even worse for the sequential algorithm which now becomes $O\left(\sum_{1}^{L} 4^{i}\right)$

Mapping the Algorithm on to the MPP

First note that we can map any arbitrary rectangle into a square be scaling one of its sides. For our purposes then we treat the input patch to the subdivision algorithm as if it were a square. At each subdivision the child patches formed are also square and have sides of length one half of their parent's side. This geometry is ideal for the array unit which is square and has a side length of 128 (a power of two). After seven subdivisions the parent patch will have created $4^7=16384$ descendants or one per MPP PE. A mapping using the vertices of the patches instead of the entire patch is not so neat and after seven subdivisions yields a 129x129 array of data values. Here we use the mapping of patches to PEs.

Child patches must be able to obtain data from their parents and also their siblings and sometimes even cousins. Since a seven level subdivision will fill each PE with a patch we will have to be able to recycle PEs which were parents in earlier generations. This may seem a lot to ask of the mesh connected communication network of the MPP but there is a straightforward and somewhat elegant solution.

Observe that all the child patches are dependent solely on the algorithm's parameters and the data of the input parent (These critters are asexual.). It is easy to initialize all the PEs with the input patch's data. Treat the entire array as one group of PEs which all represent the input patch. The first subdivision will yield four new patches each of which will also be represented by a group of PEs. Each child patch required no communication overhead to obtain parent data because, in reality, it already contained all of its parent's data. Which PEs belong to which child patch is determined by an id contained in each PE. The id corresponds to which child patch the PE represents at each level of the subdivision; after seven subdivisions the id in each PE maps it one-to-one with a single patch. Until the final subdivision, each patch is being mapped onto a group of PEs which are redundantly calculating the subdivision for that patch. The redundant calculations are occurring simultaneously so they incur no time cost. The benefit of this method, besides its simplicity, is that parent data is passed onto the children without need of the communication network.

Data routing is still used to obtain values from neighbors. Here the term neighbor is with respect to groups of PEs rather than individual ones. Since groups are largest in earlier generations, communication distances are largest for them too. During the first subdivision a routing distance across half the array is necessary for PEs of neighboring groups to communicate. At the final subdivision, adjacent PEs are group neighbors so the communication distance becomes just a single PE.

Even though the number of PE groups grows exponentially, the number of different PE types remains constant at four which correspond to the quadrants of the parent patch. Figure 2 shows how PE groups and types map onto the array. Before subdividing the first input patch, the id for each PE is determined. This is a simple process which forms seven 2-bit numbers from the concatenation of the row and column indexes already present in each PE. Figure 3 identifies the PE id types with the quadrants they correspond to. The subdivision algorithm is outlined in pseudocode below.

```
algorithm subdivide(levels:integer; h,scale:real);
```

Subdivide the parent patch defined by the values at its 4 corners to the number of levels given by "levels". H and scale are the fractal h parameter and user selected scaling factor respectively. Initializations which need be done only once per run of the algorithm such as valid = TRUE and the ID set up are assumed to have already been done and are not included in this analysis.

```
{ **** set up ***** }

      route := 128;  { # PEs on side of PE array }
      ratio := $2^{-h}$ ; { from fractal h parameter }
      std := scale; { user determined scaling parameter }

    initflag(flag); { flag shows original (input patch data)
                    where true. Used in determining the
                    calculation of the seed for the pseudo-
                    random numbers used in the $I_{stack}$ }

{ **** subdivisions **** }

      FOR level := 1 to levels DO BEGIN
            route := route div 2;
            std := std * ratio;
            center_pts(level,std);
```

```
9            side_pts(level,route,std);
10      END
```

Analysis of subdivide

The set up statements in lines 1 through 4 are executed once in constant time no matter what the value of levels. Initflag, initialises a parallel array of booleans but this uses the broadcast function and so is not dependent upon the array size. Flag is used to mark whether or not the corner data values were given as input (TRUE) or derived. Vertices marked as true forward their seed values for calculation of I_{stoch} . Flag is updated each time new data values are calculated since the new values may replace a value that was marked as TRUE in the previous generation.

Line 6 determines the routing distance between neighboring PE groups. This is halved at each subdivision until it reaches one at the final subdivision. Route is used by side_pts when obtaining information from neighboring PE groups. At line 7, std, derived from the h and scaling parameters, is adjusted for the next application in the calculation of I_{stoch} .

The lines of most interest are 8 and 9 which compute the subdivisions in two phases: center points and side points. Computation of the center points is done simultaneously throughout all PE groups without the need for shifting data. The four corner points of the parent patch are used and these are already contained within each PE. I_{det} is the average of these point values and I_{stoch} uses the corners in the selection of the pseudorandom number which is adjusted by the current value of std. The time complexity of this step will remain constant with varying levels and array sizes.

The computation of the side points is more complicated and involves shifting data in from neighboring PE groups. Since the size of a PE group varies according to what level is being processed the communication time will also vary. The algorithm as described here uses communication for the following reasons:

a. To establish if there is a valid neighbor along a particular side.
b. To gather data from a valid neighbor.
c. To transmit shared points to a neighbor.

In each of the above, neighbor refers to a neighboring PE group.

Since the calculation of side points is the only step which requires communication, we shall now shed some more details on this process. To satisfy edge constraints, interpolates of values along an outside edge (the edge of the parent patch) must depend ultimately, only on the two corner values of the parent patch's edge. When this holds patches can be matched up simply by assuring that their common edge values and generating parameters are identical. Edges not along the outside of the parent patch use values above and below and right and left to calculate their values.

The deterministic interpolate is the average of its two (when on an outside edge) or four nearest neighbors. Two of the neighbors are the vertices of the side that it lies on and the other two (when four are used) are the just calculated center points which form a perpendicular bisector of the side the interpolate lies on.

A sentinel is used to easily determine if the new vertex has a valid neighbor. Define VALID to be a parallel array of type boolean. Using the MPP broadcast instruction it is easy to set each bit of VALID contained in each PE to TRUE in constant time. Recall that attempts to shift data in from beyond the edges of the array, read in 0's (FALSE). Thus a neighbor is valid only if its VALID bit is TRUE.

Each PE group first calculates its center point then the edge interpolates. We will first discuss only the deterministic part of the calculations. The center point is trivial; it is merely the average of its four corners (plus a stochastic component). The basic algorithm for the deterministic edge components is as follows.

Note that the center point has already been computed and its value is now stored in the corners of the new child patches which lie at the center of their parent patch. There are now four different types of patches forming. The differences are based solely on the positions of the new patches with respect to their parents (see Figure 3). Each type is computed concurrently with all the others like it, thus there are four different types of edge interpolate calculations. All these are very similar and simply correspond to the particular edge being computed: top, bottom, left or right.

We can characterize the time complexity of the side point calculation step as being composed of two components:

$$T_{side\_step} = T_{calc} + T_{comm(level,array\_size)} \qquad (2)$$

In other words, the time for the side points step calculation, T_{side_step} , is made up of a nonvarying calculation portion, T_{calc} , plus a communication time, T_{comm}

. The communication time is a function of what subdivision level is being processed and of the size of the PE array. T_{comm} decreases exponentially with level and increases by the square root with array_size. Equation 2 can be rewritten as equation 3 by combining level and array_size to form the routing distance.

$$T_{side\_step}=T_{calc}+T_{comm(route)} \qquad (3)$$

If we let ROUTE be the routing distance at the first subdivision and we are using the array to process a single input patch, then ROUTE is half the number of PEs on the array side. While holding the dimensions of the array constant, the execution time of the algorithm will be a function of L, the number of subdivision levels.

Let T_{step} be the combined times for the nontime varying calculations done at each level of the subdivision. Thus T_{step} includes T_{calc} from equations 2 and 3 plus the time to calculate the center points. Execution time of the algorithm and its startup time are denoted by T_{exec} and $T_{startup}$ respectively. Assuming that the shift function is straightforwardly implemented and communication time is directly proportional to the number of bits being routed and the distance that they cover, then communication time can be expressed by a constant, K_{comm} , multiplied by the distance data must be routed (equation 4). Rewriting equation 4 to remove the summation we can derive equation 5.

$$T_{exec}=T_{startup}+\sum_{0}^{L-1}\left[T_{step}+K_{comm}*\frac{ROUTE}{2^i}\right] \qquad (4)$$

$$T_{exec}=T_{startup}+L*T_{step}+K_{comm}*(2ROUTE-1) \qquad (5)$$

Since ROUTE is a fixed value for a given machine, the time complexity is not affected by it or the other constant terms. The execution time complexity of the algorithm, denoted $Comp(T_{exec})$, is directly proportional to L, the number of subdivision levels. The complexity is log the number of final patches produced and, due to the data dependencies inherent in the algorithm, this is the best result that can be expected.

$$Comp(T_{exec})=L \qquad (6a)$$

$$Comp(T_{exec})=log(number\ of\ final\ patches) \qquad (6b)$$

To be able to subdivide more levels using the same algorithm, larger arrays of PEs must be built. (It should be noted that the 128x128 array size of the MPP is a reasonable size for graphics applications which are still typically 512x512 pixels per screen.) If we allow the array size to grow and continue the algorithm to the maximum level, L_{max} , that the array size will allow, we can compute L_{max} from ROUTE (equation 7).

$$L_{max}=\log_2(ROUTE)+1 \qquad (7)$$

Using equation 7 and substituting into equation 5 with $L=L_{max}$ we obtain equation 8. This gives the execution time as a function of L when the architecture of the machine varies to always have one PE per patch after the final subdivision.

$$T_{exec}=T_{startup}+L*T_{step}+K_{comm}*(2^L-1) \qquad (8)$$

As L grows large, the third term begins to dominate. But the number of final patches being produced is growing as 4^L . Taking the limit as L grows large shows that the parallel algorithm is still substantially better than the sequential one (equation 9). Recall from earlier analysis that the sequential time algorithms must be at least $O(4^L)$.

$$\frac{2^L}{4^L}=\frac{2^L}{2^{2L}}=\frac{1}{2^L}\rightarrow\ 0\ as\ L\rightarrow\infty \qquad (9)$$

The time complexity can be decreased below L if the data dependencies can be eliminated. This is true if multiple input patches can be processed concurrently. Instead of embedding a single patch in the array, begin by embedding four or sixteen, etc. Each patch is processed concurrently using a ROUTE which corresponds to the number of PEs it covers. Slight modifications to the identifier labels and the data sentinel for boundary detection would be required to the basic algorithm.

Conclusions

A parallel algorithm to subdivide rectangular patches using stochastic interpolation was developed. The algorithm was designed for mesh connected SIMD computers and was implemented on the MPP at Nasa Goddard Space Flight Center. For a fixed architecture SIMD, the algorithm has time complexity of log the number of final patches produced; this is the best that can be expected due to the data dependencies imposed.

Timing data was collected for a nonoptimized version of the algorithm using parallel pascal on the MPP. Approximately 30 milliseconds of processing time is required to subdivide one patch seven levels into 16384 final patches. This figure compares favorably to the 60 milliseconds required on the special purpose STINT processor (Ref. 8). Figure 4, Color Plate VI shows the evolution of the algorithm as it produces a 512x512 pixel "sky" from 16 input patches. Figure 5, Color Plate VII shows a scene composed of textures generated similarly but using different parameter values and color mappings.

A similar algorithm which recursively subdivides triangles (Ref. 2) also maps well to SIMD mesh connected machines and is detailed in (Ref. 11). Parallel machines such as the MPP, besides running existing paradigms faster allow insights into ways they may be expanded. A direction for future research is how the generating parameters of this stochastic interpolation algorithm may be increased to higher dimensions to make the generic algorithm more powerful.

References

1. *Fundamentals of Interactive Computer Graphics*, Foley, J. D. and van Dam, A., Addison-Wesley Reading MA, 1983
2. Computer Rendering of Stochastic Models, Fournier, A., Fussell, D., Carpenter, L., Comm ACM, 25(6), pp. 371-384, Jun 82
3. Frame Buffer Algorithms for Stochastic Models, Fournier, A., Milligan, T., IEEE CG & A, 5(10), pp. 40-46, Oct 85
4. Using Stochastic Modeling for Texture Generation, Haruyama,Shinichiro, Barsky, B., IEEE CG & A, 4(3), pp. 7-19, Mar 84
5. Fractional Brownian Motions, Fractional Noises and Applications, Mandelbrot, B.B., Van Ness, J.W., SIAM Review, 10(4), pp. 422-437, Oct 68
6. The Fractal Geometry of Nature Rev. ed., Mandelbrot, B.B., W.H. Freeman and Company, San Francisco,1983
7. *Algorithms for Graphics and Image Processing*, Theo Pavlidis, Computer Science Press, Rockville MD, 1982, pp.1-4
8. A Hardware Stochastic Interpolator for Raster Displays, Piper, T.S. Fournier, A., Comp Graphics, 18(3), pp. 83-92, Jul 84
9. Early Vision: From Computational Structure to Algorithms and Parallel Hardware, Tomaso Poggio, *Computer Vision, Graphics, and Image Processing 31:* 139-155 (1985)
10. Ray Casting for Modelling Solids, Scott D. Roth, *Computer Graphics and Image Processing 18(2):* 109-144, Feb. 1982
11. Generating Fractal-like Surfaces on General Purpose Mesh-Connected Computers, Wainer, M., submitted to IEEE Trans on Comp., Jan 86

Acknowledgments

Mr. Wainer's work was made possible by a NASA Graduate Student Fellowship, Training Grant NGT 02-002-800, and on-site sponsorship and assistance of Dr. James Strong. Dr. McAnulty's access to the MPP was made easier through the cooperation of Dennis Gallagher, ES-01, Marshall Space Flight Center, who enabled direct access to the SPAN network at the Space Science Laboratory.

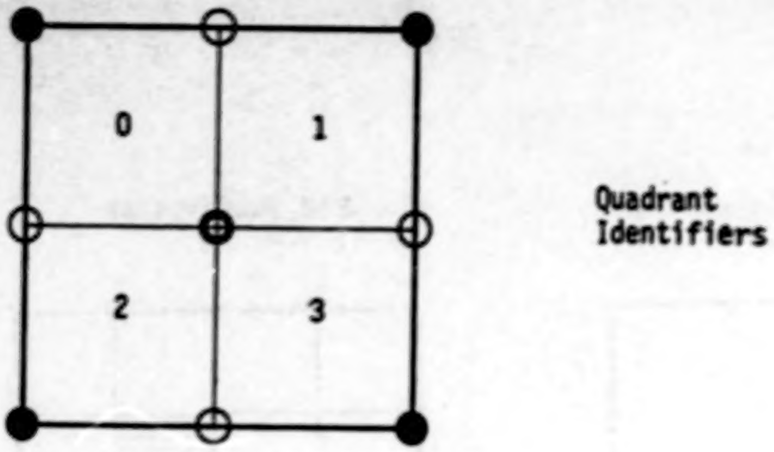

0
1
2
3
Quadrant Identifiers

Figure 3: Quadrants and their type designations

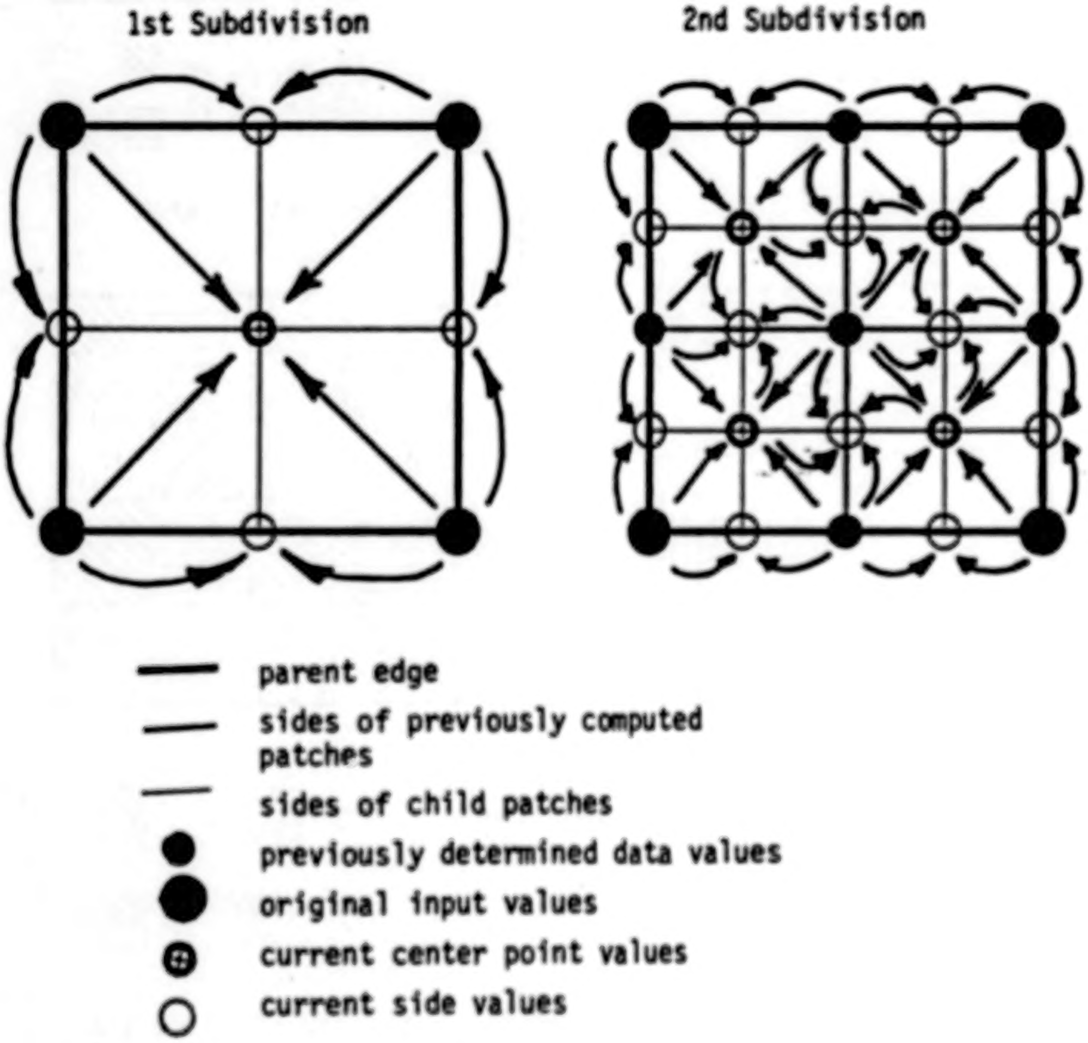

1st Subdivision
2nd Subdivision
parent edge
sides of previously computed patches
sides of child patches
previously determined data values
original input values
current center point values
current side values

Figure 1: A Graphical Description of the Subdivision Algorithm

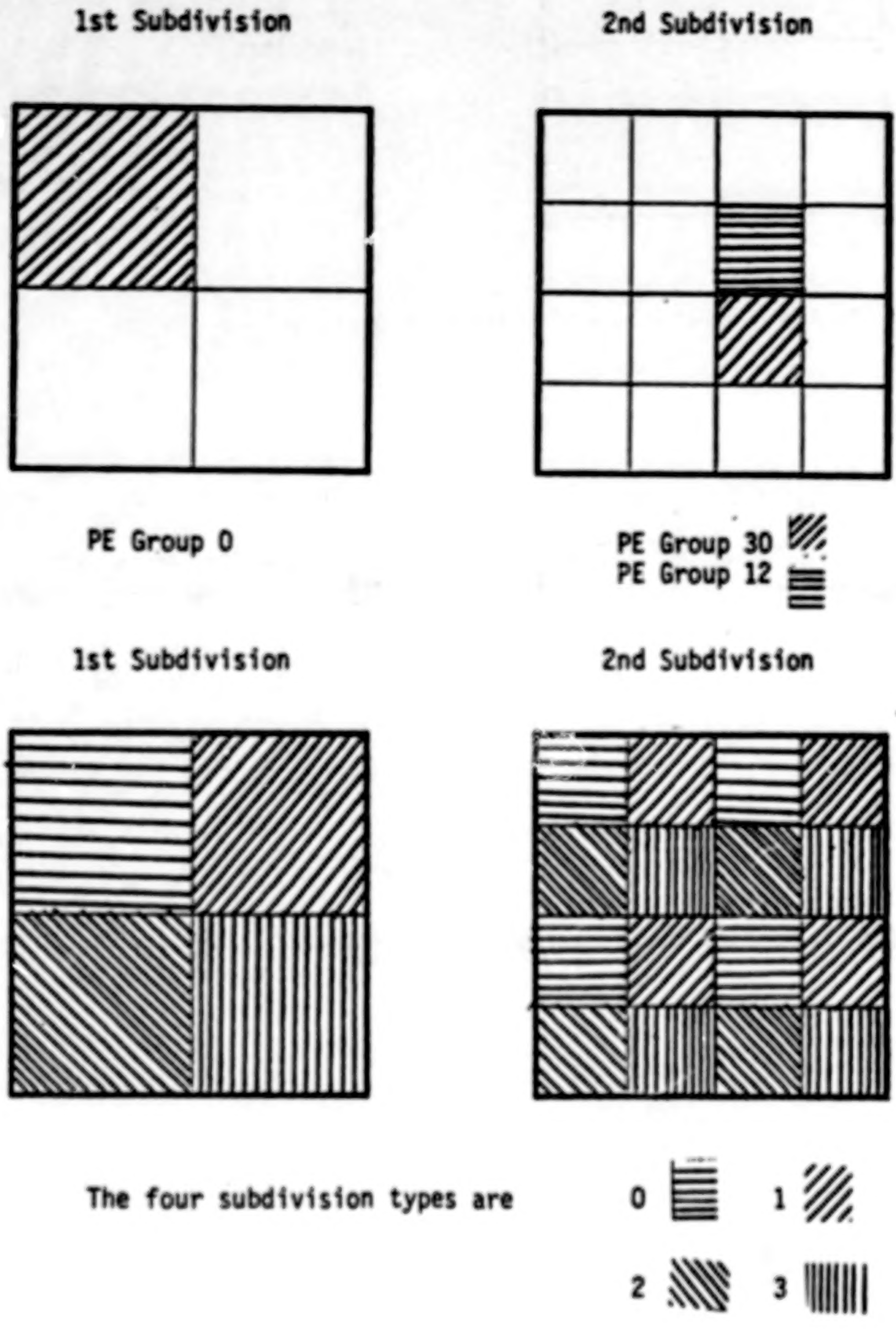

1st Subdivision
2nd Subdivision
PE Group 0
PE Group 30
PE Group 12
1st Subdivision
2nd Subdivision
The four subdivision types are
0
1
2
3

Figure 2: Examples of PE groups and types mapped on to the array. A group corresponds to a child patch. A type corresponds to particular quadrant of the parent patch.

INVITED PAPER

Dr. Rolf D. Fiebrich
Thinking Machines Corporation

N87-26560

Reverse Time Migration - A Seismic Processing Application on the Connection Machine

Rolf-Dieter Fiebrich
Thinking Machines Corporation
Cambridge, MA 02142

Abstract

This article describes the implementation of a reverse time migration algorithm on the Connection Machine, a massively parallel computer. Essential architectural features of this machine as well as programming concepts are presented. The data structures and parallel operations for the implementation of the reverse time migration algorithm are described. The algorithm matches the Connection Machine architecutre closely and executes almost at the peek performance of this machine.

~~PRECEDING PAGE BLANK NOT FILMED~~

PRECEDING PAGE BLANK NOT FILMED

N87-26560

Introduction

This paper describes the implementation of a reverse time migration algorithm on a massively parallel computer. These computers, with thousands of processing elements are entering the marketplace and offer better cost performance than conventional mainframes, and more importantly, promise to reach significantly higher absolute performance levels in the coming years than those which can be realised by conventional architecures. This paper discusses how this technology can be utilised efficiently for extremely computation-intensive algoritms in seismic processing.

This paper focuses on the implementation of a reverse time migration algorithm for 2D seismic processing. The close match of this application with the Connection Machine architecture results in substantial speedups compared to conventional mainframes and strongly suggests that this machine puts 3D seismic processing in reach.

The remainder of this paper summarises the architecture and programming issues of massively parallel computers, followed by a discussion of the reverse time migration algorithm and its implementation. A brief performance summary for the program follows. The paper concludes with comments on what massively parallel computers can do today for seismic processing problems and the promise of this technology for the future.

Architecture and programming of a massively parallel machine

A massively parallel computer can be viewed as a machine which can operate on thousands of data objects at once, whereas a conventional computer operates on one data object at a time. If an application permits for instance to operate on all elements of a large vector or matrix or on all nodes or edges of a graph in parallel, then substantial execution speed improvements can be obtained if a large number of processors is available, ideally one processor for each data object.

A program for such a computer looks very much like a program for a conventional computer, except that certain program variables are declared to be parallel variables which means that operations on those variables can take place in parallel. Instead of using a loop statement to process all the elements of a vector or a matrix or a graph, one uses a select statement for all the elements and all statements in the body of the select statement are performed in parallel on the selected set, like the statements in the body of a loop statement are performed on each element visited.

The above sketches an idealised programming model. Physical parallel machines usually have some limitations for implementing this model efficiently. For instance, processors have to be assigned to data objects at compilation time and cannot be reassigned during execution time. Also the computation which a processor performs an a data object that is assigned to it in general involves access to data ojbects assigned to other processors. The execution time of "local access" vs. "nonlocal" access is of course different. Programmers need to take this into account for writing efficient programs, but this is not much different from optimisation considerations on conventional computers.

A physical computer which implements this model is the Connection Machine [Hillis, 1985]. Most of the experience reported in this paper was gained on this machine. The Connection Machine uses a conventional host which provides all the infrastructure for program development and communicating with other computers. This host has, however, several important enhancements. It has a significantly enlarged memory which is partitioned in equal chunks and each chunk has a processor associated with it. These processors cannot just access data in their part of the memory, but can also access the entire memory of the machine. All processors can perform these accesses in parallel. The processors of the Connection Machine can be viewed as an extension of the execution unit of the host. Figure 1 illustrates the architecture of the Connection Machine.

The Connection Machine model which was used for most of the work described in this paper has a maximum of 32 MBytes of memory and 64 K processors. The number of processors which a programmer sees is typically significantly larger than the number of physical processors. The system supports a virtual processor concept. The host for the Connection Machine can be either a VAX or a Lisp Machine. Parallel programming concepts as described above are implemented as straigtforward extensions of C and Lisp.

Reverse time migration

Finite-difference scheme

The reverse time migration process is well known and well documented in the literature [McMechan,1983]. Conceptually, reverse time migration, as with all depth migrations, involves the transfer of data from the

$(y, z = 0, t)$ time plane to the $(y, z, t = 0)$ depth plane. This concept is illustrated graphically in Figure 2. For the acoustic case, which is discussed here, wave propagation through the earth is governed by the acoustic wave equation. Attention is further restricted to the two dimensional case. Hence, the wave equation has the form

$$U_{yy} + U_{zz} = \frac{1}{v^2(y, z)} U_{tt} \tag{1}$$

where U is the acoustic wave field. Reverse time migration is based on an exploding reflector concept wherein the interface between rock strata explode with sound at time t_0. From that moment on waves propagate according to the above wave equation at velocities one-half their true velocity in accordance with the exploding reflector model. If acoustic measurements are made at various places along the earth's surface for all subsequent time we have the equivalent of a zero offset stacked section. Migration is implemented by reversing the process and exciting mesh points at $z = 0$ with the time reversed recorded signals. Since the wave equation is ambivalent to the direction of time this is no problem. The recorded signals act as boundary values in the numerical solution of the wave equation.

Discretisation of the acoustic wave equation in time and space follows traditional numerical methods [Dablain,1986]. Using these methods equation (1) may be approximated by a fourth order spatial and second order temporal operator. The notation

$$U^k_{i,j} = U(y_i, z_j, \hat{t}_k)$$

is used in writing the difference operator (where $\hat{t}_k$ refers to reverse time) as

$$\begin{aligned} U^k_{i,j} &= 2U^{k-1}_{i,j} - U^{k-2}_{i,j} \\ &+ \frac{A^2}{12}[\,16(U^{k-1}_{i+1,j} + U^{k-1}_{i-1,j} + U^{k-1}_{i,j+1} + U^{k-1}_{i,j-1}) \\ &\quad - 60U^{k-1}_{i,j} - U^{k-1}_{i+2,j} \\ &\quad - U^{k-1}_{i-2,j} - U^{k-1}_{i,j+2} - U^{k-1}_{i,j-2}\,] \end{aligned} \tag{2}$$

where

$$A = v(y, z)\frac{\delta t}{\delta y}.$$

For purposes of simplifying the explanation of the implementation it will be useful to have the simpler second order spatial operator as well. The difference equation is [McMechan,1983]

$$\begin{aligned} U^k_{i,j} &= 2(1 - 2A^2)U^{k-1}_{i,j} - U^{k-2}_{i,j} \\ &+ A^2[\,U^{k-1}_{i+1,j} + U^{k-1}_{i-1,j} \\ &\quad + U^{k-1}_{i,j+1} + U^{k-1}_{i,j-1}\,]. \end{aligned} \tag{3}$$

Consider equation (3). There are three time steps involved and three spatial points in each direction. This equation is illustrated graphically in Figure 3 where the three time steps are represented as three depth planes in reverse time, $\hat{t}_i$. Each illustrated plane is a small part of a larger mesh on which the finite-difference scheme is carried out. The data values required to compute the current grid point value, $U^k_{i,j}$, are identified as the grid point's own previous and second previous value, and its immediate neighbors' previous value. Again, refer to Figure 3 for a graphical representation of the process. Initially, the previous and second previous depth planes are zero. This corresponds to the assumption that all signals are recorded until they are identically zero. Conceptually the reverse time migration proceeds as follows; 1) load the boundary value corresponding to time step $\hat{t}_0$, 2) compute all the grid points in the current depth plane, 3) push the stack of depth planes so that the current plane becomes the previous plane and the previous plane becomes the second previous plane, 4) repeat until time $\hat{t}_{max}$ (or t_0) is reached. The final solution will be an acoustic wave field reconstruction of the exploding reflectors imaged at time t_0 for all depths.

Because of reflections from the boundary of the computational grid it is desirable to implement absorbing boundary conditions along the two edges and the bottom [Clayton,1977]. When the wave field in the depth plane is computed in step (2) above, an absorbing boundary difference scheme must be used on the edges.

Parallel Implementation

Finally, we are ready to discuss the paralle algorithm for reverse time migration. We assign a processor to each grid point in the finite-difference mesh Figure 3. To compute the current value in a processor requires that the processor reference its own local memory for the previous and second previous value. It also requires that the processor get the previous value from each of its neighbors. This is exactly what is done in mapping the algorithm onto the Connection Machine. The time axis in Figure 3 corresponds to the memory axis of each processor. The time section is usually larger, in terms of the number of samples per trace, than the depth section. As a result, the time section is incrementally fed into the Connection Machine.

In generating a new data value in each processor (at each grid point) two of the memory accesses are local and the rest are non-local. The non-local accesses require utilisation of the general communication system. Four such accesses are needed for the second order finite-difference operator. In addition, to load the boundary value at the beginning of each time step requires another non-local memory access. To implement the absorbing boundary conditions the processors on the edge of the computational grid are selected and use an absorbing boundary finite-difference operator to compute a new value. Consider a typical unmigrated seismic section. There might be 1024 traces and 2500 time samples. If 512 depth steps are desired, there must be 512K processors using the purposed parallel approach. Since there are only 64K processors, virtual processors must be used for almost all practical cases. Using a virtual processor ratio of 8:1 will provide the required 512K processors.

Timing

The total execution time for a data set of the size 1425 x 625 x 2500 is 441 seconds. This computation takes several hours on a large mainframe.

The whole issue of timing is obviously machine specific and is instantly out of date due to hardware improvements. The point is that parallel computers can compete with the fastest serial supercomputers. In addition, the very fact that the Connection Machine has floating point instruction times measured in hundreds of microseconds instead of tens of nanoseconds points provides significant technological improvement.

Conclusions

Results from the reverse time migration implementation and from the many other non-seismic applictions that have been programmed indicate that massively parallel architectures are viable and can perform at supercomputer levels.

In the specific case of a reverse time migration algorithm, performance improved by two orders of magnitude relative to a VAX 785. This improvement is achieved despite relatively simple individual processors in the fine-grain computer. The fact that there are 64K such processors far outweighs the fact that each processor is slow. Overall, vast speed improvements are possible both for reverse time migraion in particular and for seismic processing in general. One exciting possibility is that the dream of interactive interpretation/processing might be realised. Imagine a work station where an interpreter can repeatedly migrate a section, trying different velocity models each time. In so doing, the iterative process of converging on a satisfactory depth model might take a few hours instead of many days. In addition, because the interpreter would be intimately involved in the processing, the final result would be better than with batch processing. This is just one computational problem that parallel computers can address.

Acknowledgements

The author wishes to thank Rob Fricke and Monica Wong for the implementation of the reverse time migration algorithm on the Connection Machine. Many thanks go also to Olivia Fiske for her quick help in preparing this document.

References

[Clayton,1977] Clayton, R. W., and Engquist, B., 1977, Absorbing boundary conditions for acoustic and elastic wave equations: Bulletin fo the Seismological Society of America, 67, 1529-1540.

[Dablain,1986] Dablain, M. A., 1986, The application of high-order differencing to the scalar wave equation: Geophysics, 51, 54-66.

[Hillis,1985] Hillis, W. D., 1985, The Connection Machine: M.I.T. Press.

[McMechan,1983] McMechan, G. A., 1983, Migration by extrapolation of time-dependent boundary values: Geophysical Prospecting 31, 413-420.

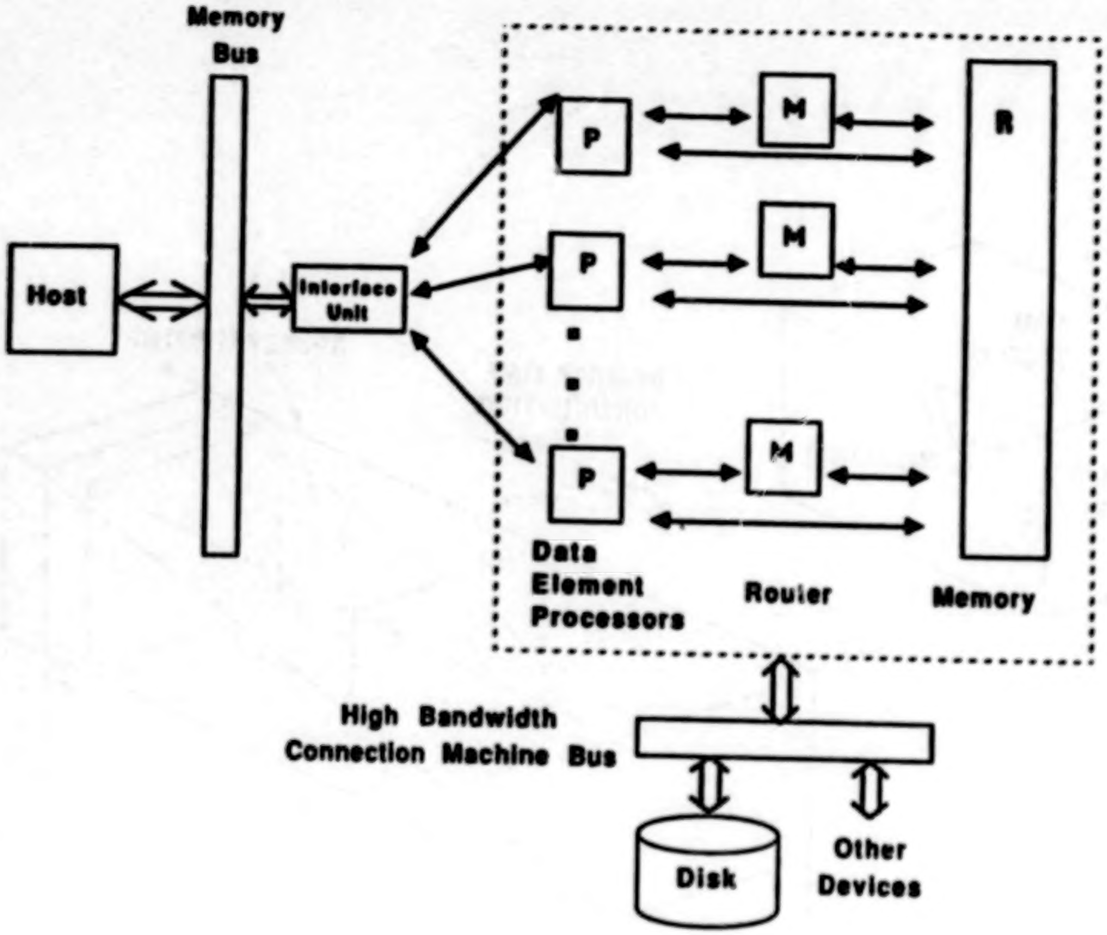

Memory Bus
Host
Interface Unit
P
P
P
M
M
M
R
Data Element Processors
Router
Memory
High Bandwidth Connection Machine Bus
Disk
Other Devices

Figure 1: Architecture of the Connection Machine System

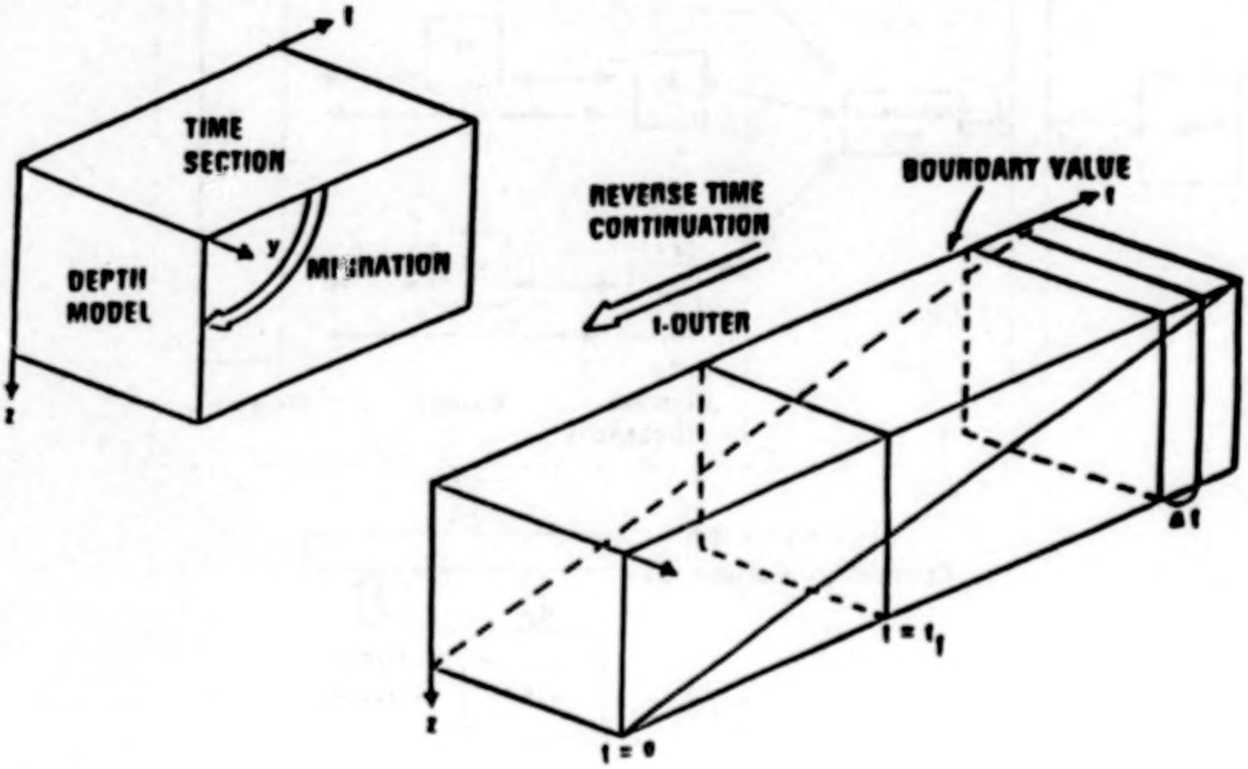
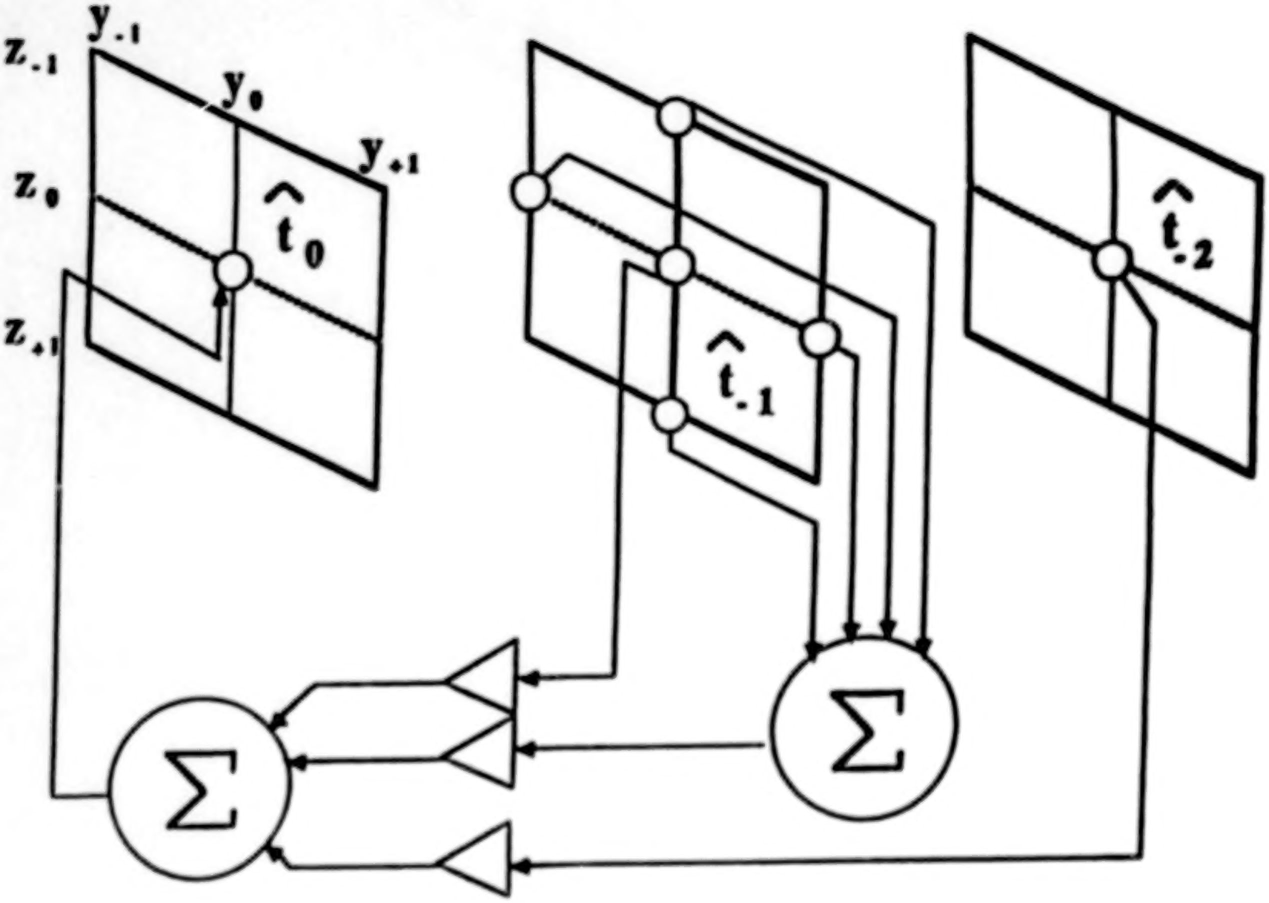

t
TIME SECTION
y
DEPTH MODEL
MIGRATION
z
REVERSE TIME CONTINUATION
t-OUTER
BOUNDARY VALUE
t
Δt
t = t_1
z
t = 0

Figure 2: Reverse Time Migration



y_{-1}
y_0
y_{+1}
z_{-1}
z_0
z_{+1}
$\hat{t}_0$
$\hat{t}_{-1}$
$\hat{t}_{-2}$
Σ
Σ

Figure 3: Computation of Second Order Difference Operator

COMPUTER SCIENCE SESSION II

Session Chair: Prof. Harold M. Hastings
Hofstra University

PRECEDING PAGE BLANK NOT FILMED

N87-26561

SOLVING VERY LARGE, SPARSE LINEAR SYSTEMS ON MESH-CONNECTED PARALLEL COMPUTERS

Torstein Opsahl[*]

MRJ, Inc.
Perkin-Elmer Advanced Development Center
Oakton, Virginia 22124

John Reif[**]

Duke University
Durham, North Carolina 27706

ABSTRACT

In this paper, the implementation of Pan and Reif's Parallel Nested Dissection algorithm on mesh-connected parallel computers is described. This is the first known algorithm that allows very large, sparse linear systems of equations to be solved efficiently in polylog time using a small number of processors. We describe how the processor bound of PND can be matched to the number of processors available on a given parallel computer by slowing down the algorithm by constant factors. Also, for the important class of problems where G(A) is a grid graph, we detail a unique memory mapping that reduces the inter-processor communication requirements of PND to those that can be executed on mesh-connected parallel machines. The paper concludes with a description of an implementation on the Goodyear Aerospace Massively Parallel Processor (MPP), located at NASA Goddard Space Flight Center, for which we give a detailed discussion of data mappings and performance issues.

Keywords: Parallel Computation, Parallel Nested Dissection, Recursive Factorization, Mesh-connected Network, Separator, Grid Graph.

INTRODUCTION

The solution of large, sparse linear systems, Ax=b, pervades many areas of physics and engineering. Although parallel algorithms for tackling these problems have existed for a number of years, they have usually been impractical to implement because of unrealistically high time bounds or processor bounds, or both. In other cases, numerical stability has been a problem; that is, unless the calculations were performed in infinite precision they would yield no solution at all. However, none of the above problems apply to Pan and Reif's (Ref. 8) parallel nested dissection (PND) algorithm. This algorithm is based on computing a special recursive factorization of A, thus reducing the problem of inverting a large sparse matrix to that of inverting a number of much smaller dense matrices. PND has a considerably smaller processor bound than other polylog parallel methods for solving sparse linear systems. Furthermore, for any given parallel computer with p processors, the algorithm can be slowed down to give the best known time bound for this processor bound p.

If A is an nxn matrix, we define a graph G(A)=(V,E) to be the undirected graph with vertex set V={1,...,n} and edge set $E=\{(i,j)|A_{ij}$ not equal to 0}. Here we shall confine our attention to the application of PND to sparse, linear systems where G(A) is a two-dimensional (2-D) grid graph. Such systems occur very extensively in the solution of partial differential equations. However, it should be emphasised that the algorithm is not restricted to this class of problems.

[*] Previous association, Science Applications Research, Lanham, Maryland.

[**] This work was supported by Office of Naval Research Contract N00014-80-C-0647 and NSF grant DCR-850351

PRECEDING PAGE BLANK NOT FILMED

The Massively Parallel Processor (MPP) is an SIMD controlled, fine-grained, 2-D mesh-connected parallel computer with 16,384 Processing Elements (PEs) built by Goodyear Aerospace for NASA. This paper will describe how PND can be implemented on this computer and other parallel machines with a similar architecture. The implementation is particularly challenging because of the SIMD control structure of these computers. Central to the implementation is a unique memory mapping that reduces the communication requirements of PND to those that can be executed using only mesh connections. Also, the algorithm has been implemented on the Connection Machine, built by Thinking Machines Corporation, whose processors are connected in a hypercube. This implementation is described in Ref. 5.

In the next section, we give a more detailed description of PND, while the final section contains the implementation method for mesh-connected parallel computers together with some performance estimates.

DESCRIPTION OF PND FOR GRID GRAPHS

The method of parallel nested dissection (PND) and its proof has been described in detail elsewhere (Ref. 8). Here, we will give a relatively self-contained overview of the method. Since it is far from obvious how PND can be used on the MPP, the description concentrates on those aspects of the method that are critical to understanding our implementation for mesh-connected parallel computers.

Fundamental to the nested dissection of undirected graphs is the idea of separators. A separator is a set of vertices that partitions a graph into two sub-graphs that are connected only through the separator. Each of the sub-graphs must contain no more than 2/3 and no less than 1/3 of the nodes of G(A). For a graph of n vertices, the size of any separator is bound by a function s(n). In the case of planar graphs s(n) is $O(n^{1/2})$. An undirected graph is said to have an s(n)-separator family if the class of all its sub-graphs has an s(n)-separator family. Binary trees, for instance, have a 1-separator family while a d-dimensional grid of uniform size in each dimension has $n^{1-(1/d)}$ -separators.

The removal of separators from a graph, G(A), and its resultant sub-graphs, is in fact analogous to eliminating the unknowns of the associated linear system, in the order given by the numbering of the vertices in the separators. Sequential methods for solving sparse, linear systems have long made use of nested dissection (Ref. 3) to reduce the problem to that of inverting dense matrices of size at most s(n)xs(n). PND is the first algorithm for efficiently solving sparse, positive definite linear systems, Ax=b, on parallel computers that only requires polylog time and $s(n)^3$ processors. Moreover, the algorithm is numerically stable. PND constructs in O(logn) stages a special recursive factorization of A. This factorization is distinct from the LDL^T-factorization used in sequential dissection. Another important difference between sequential dissection and PND is that the latter requires the elimination of many, rather than one, separators at each step of the algorithm.

Suppose G(A) has a separator that decomposes the graph into two sub-graphs. Furthermore, suppose that these sub-graphs can be partitioned recursively through the use of separators. In this way we can construct a separator tree (see figure 1) whose root is the separator of G(A). This root has two children that are the separators of the two sub-graphs. Each of these children is in turn the parent of two new children and so on. The leaves of the separator tree are singleton node sets of remaining sub-graphs.

One of the general difficulties of PND is to compute efficiently in parallel s(n)-separators. Fortunately, for the practically important case of grid graphs, which we are concerned with here, there is a simple way to do this in O(logn) steps. At the completion of these steps the vertices of the graph have been renumbered from 1 to n using the separator tree as a guide. PND requires that A is permuted to reflect this renumbering. Assuming this has been done, Pan and Reif (Ref. 8) define a recursive s(n)-factorization by a sequence of matrices A_0, $A_1, \ldots, A_d$ where,

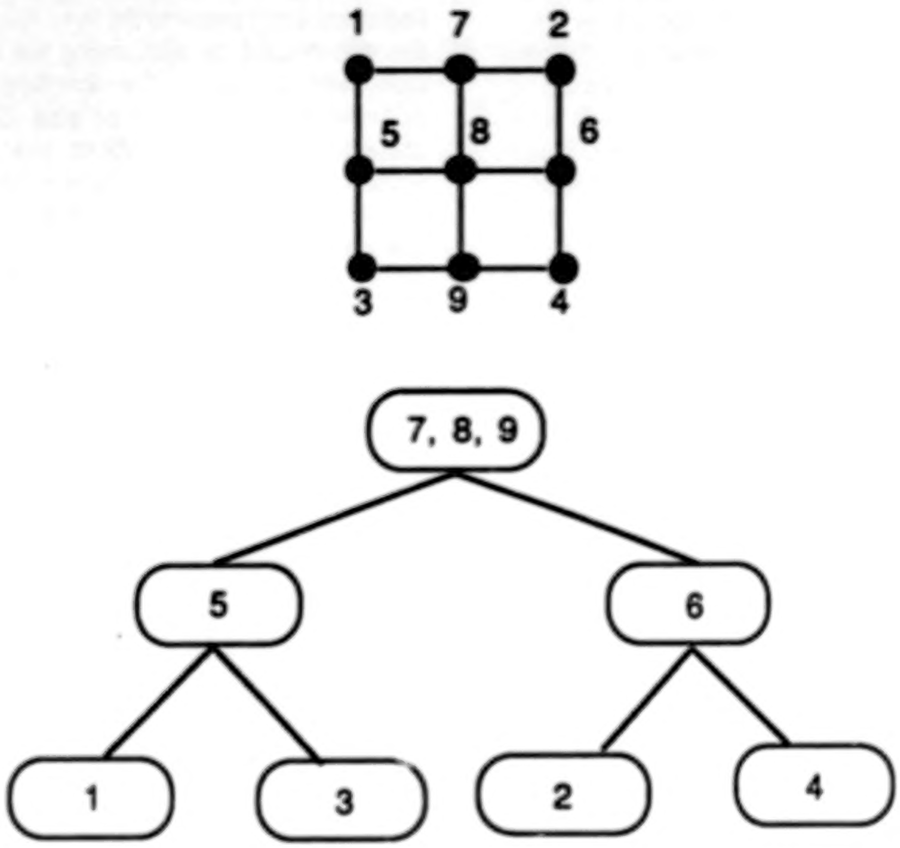


Figure 1 - Separator tree for a 3x3 grid graph

$A_0 = PAP^T$; P is a permutation matrix

$$A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix} \qquad (1)$$

$$Z_h = A_{h+1} + Y_h X_h^{-1} Y_h^T; \; h=0,1,\dots,d-1$$

There are several important points to note about equation (1). Let d be O(logn). Also, let N_h be the number of separators at level h and $n_{h,k}$ ($k= 1 \dots N_h$, $n_{h,k} <= (2/3)^h$) the maximum size of any separator at the h^{th} level. It can then be seen that X_h is a block-diagonal matrix consisting of N_h square blocks of sizes at most $n_{h,k} x n_{h,k}$. Thus, X_h^{-1} can be computed by inverting these small diagonal blocks separately. It also follows that the matrix product $Y_h X_h^{-1} Y_h^T$ can be decomposed into $O(N_h)$ triplets of sizes $O(n_{h,k} x n_{h,k})$. Pan and Reif (Ref. 8) prove that both of the above operations can be performed in $O(\log^2 s(n))$ time using at most $s(n)^3$ processors. In fact, the main advantage of the recursive s(n)-factorization is that it only requires O(logn) stages. This implies that the entire s(n)-factorization of A can be computed in$O(\log n(\log^2 s(n))$ time using $s(n)^3$ processors. However, on even the most parallel of currently available computers such as the MPP and the Connection Machine, this bound will severely restrict the size of problems that can be solved. Fortunately, the PND algorithm can easily be slowed down so that the number of processors required matches that of a specific parallel machine. In particular, if $s(n)=n^{1/2}$ then n processors is sufficient provided the algorithm's speed is reduced by an $n^{1/2}$ factor. As a matter of fact, a careful time analysis shows that the total time to complete the s(n)-recursive factorization of A in this case is $O(n^{1/2})$. As we shall see in the next section this allows very large sparse matrices to be inverted on the MPP.

Finally, the recursive s(n)-factorization allows us to write

$$A_h = \begin{bmatrix} I & 0 \\ Y_h X_h^{-1} & I \end{bmatrix} \begin{bmatrix} X_h & 0 \\ 0 & A_{h+1} \end{bmatrix} \begin{bmatrix} I & X_h^{-1} Y_h^T \\ 0 & I \end{bmatrix} \quad \text{--- (2)}$$

and hence

$$A_h^{-1} = \begin{bmatrix} I & -X_h Y_h^T \\ 0 & I \end{bmatrix} \begin{bmatrix} X_h^{-1} & 0 \\ 0 & A_{h+1}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -Y_h X_h^{-1} & I \end{bmatrix} \quad \text{--- (3)}$$

Thus, given an s(n)-factorization of A, it is easy to recursively compute $A^{-1}b$ for any column vector b of length n. This "backsolving" computation can be performed in time O(lognlogs(n)) using $s(n)^2$ processors. It should be noted that here we do not have to slow down the parallel computation when $s(n)=n^{1/2}$ and the number of available processors is n.

In summary, a step in the recursive s(n)-factorization of A is accomplished by moving one level up in the separator tree removing the level h separators from the sub-graphs by eliminating the corresponding unknowns of A_{h+1}. This involves $O(N_h)$ matrix operations on matrices of size $O(n_{h,k}^2)$ (see above). If these operations are performed by systolic algorithms, they require a maximum of $s(n)^2$ processors and take O(s(n)) time. Once the root of the tree is reached ($N_h=1$, $h=d-1$), the s(n)-factorization is complete. Two traversals of the separator tree are then necessary to perform the "backsolving" because of the recursive nature of equation (3) and the total time for these computations is O(lognlogs(n)).

IMPLEMENTATION OF PND ON THE MPP

In this section we will discuss an implementation of PND, for the case where G(A) is a $n^{1/2} x n^{1/2}$ grid graph, for mesh-connected parallel computers, in general, and for the MPP in particular.

Representing a Grid Graph as a Set of Boxes

To be able to fully explain our implementation of PND on the MPP, we first need to expand on some of the ideas of the previous section. Consider a phase h of the recursive factorization where X_h is a block-diagonal matrix consisting of a total of N_h blocks $x_{h,k}$ ($k=1..N_h$) of size at most $n_{h,k} x n_{h,k}$. Each of these diagonal blocks can be associated with a given separator. The same applies to the triplets $y_{h,k}(x_{h,k})^{-1}(y_{h,k})^T$ (size $O(n_{h,k}^2)$) of $Y_h X_h^{-1} Y_h^T$. Furthermore, their existence implies that Z_h can be broken up into N_h blocks $z_{h,k}$ also of size $O(n_{h,k}^2)$. This in turn means that A_h can be considered as being made up of N_h smaller matrices $a_{h,k}$. Each of these will consist of a $x_{h,k}$, $y_{h,k}$, $(y_{h,k})^T$ and a $z_{h,k}$. The crucial observation, however, is that the $m_{h,k}$'s (m= a, x, y, z) are much smaller and denser than the big matrices to which they belong. Thus, a key element of the MPP implementation of PND is to compute A_{h+1} as N_{h+1} a_{h+1}'s. We will now describe in detail our implementation of the h^{th} stage of PND where A_{h+1} must be computed from A_h.

In figure 2, the grid graph of figure 1 has been extended so that each vertex is the junction of four edges, each of unit length. The resultant graph may be represented as a tiling of 2-D boxes with the

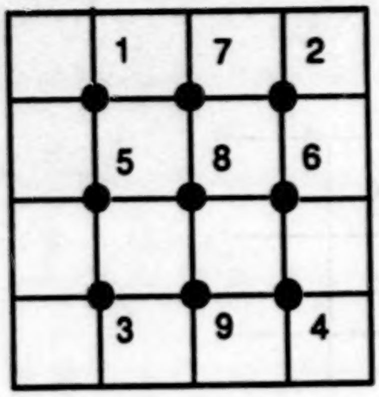


Figure 2 - 3x3 grid graph

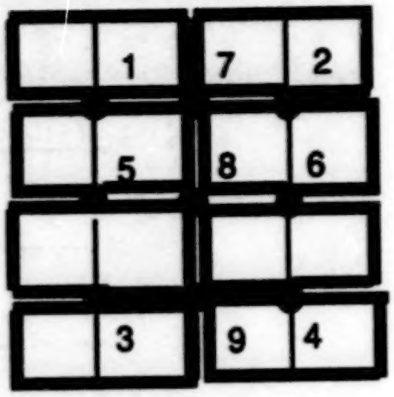


Figure 3 - Box representation of grid graph

vertices lying along their perimeters (see figure 3). Each box is a $2^h x 2^{h+1}$ rectangle or a $2^h x 2^h$ square depending on whether h is odd or even, respectively. Now, in computing the $a_{1,k}$'s (h= 0), the level zero separators are removed (vertices 1, 2, 3 and 4 for the grid graph of figure 3). This may be viewed as merging pairs of boxes into single boxes. All vertices strictly internal to a box will be eliminated by the s(n)-factorization of A. The box representation of a unit edge length grid graph can be used at all levels in the separator tree and its traversal therefore translates into repeated box-compositions. A box $b_{h,k}$ has $2(n_{h,k} + n_{h-1,k} + 2)$ vertices on its perimeter. A matched pair of level h boxes is defined as two boxes that have a common separator $s_{h,k}$. The first vertex on either side of a separator will be referred to as contact points denoted by c_0 and c_1, respectively.

The arithmetic associated with the box-composition breaks down as follows: To compute the $a_{h+1,k}$'s will require N_h dense matrix inversions (or LDL^T decompositions), $2N_h$ matrix multiplications and N_h matrix subtractions. These operations will be performed by systolic algorithms using sub-arrays of $O((n_{h,k})^2)$ processors (see below for alternative non-systolic algorithms for the MPP). Because the two $a_{h,k}$'s of a pair of boxes are not completely disjoint due to the boxes' common edges it will also be necessary to merge pairs of $a_{h,k}$'s by adding together the common edge coefficients in order to compute the $z_{h,k}$, $y_{h,k}$, $x_{h,k}$ and $y_{h,k}^T$ needed in the calculation of $a_{h+1,k}$. This requires another N_h matrix additions. It should be noted that the final stage (h=d-1) of the s(n)-factorization of A involves the inversion of a single matrix of size s(n)xs(n) and that the time required for this is O(s(n)). Hence, the cost of the final stage is the dominant element in the total time for the factorization.

The arithmetic described above will require inter-processor communications. The patterns of these, as well as their cost, depend on how A and its factors are mapped onto the local memories of the PEs. In the following a unique memory mapping scheme is described that creates local communication patterns that are supported by mesh-connected machines such as the MPP. This scheme also ensures that data can be sent to the correct processors using SIMD control so that no explicit address calculations are required.

Mapping a Box Representation of a Grid Graph onto the MPP.

Let each box of a box representation of a grid graph correspond to a $(n_{h,k}+ n_{h-1,k}+ 2)^2$ neighborhood of processors such that the coefficients of adjacent boxes in the graph will be stored in adjacent neighborhoods of PEs. Within each of these neighborhoods, data is laid out using the following rule: The ordering of the coefficients should be that given by taking the vertices of the perimeter of the box in a clock-wise direction, starting in the lower left hand corner, just above the corner vertex. Figure 4 shows the result of using our memory mapping scheme for the pair of boxes of figure 3 with

PE : i ->

j ↓

North Box

	(-)	(-)	(-)	(7)	(1)	(-)
(-)	0	0	0	0	X	0
(-)	0	0	0	0	X	0
(-)	0	0	0	0	0	0
(7)	0	0	0	X	X	0
(1)	0	0	0	0	0	0
(-)	0	0	0	0	0	0

South Box

X - non-zero coefficient

	(-)	(1)	(7)	(8)	(5)	(-)
(-)	0	0	X	0	X	0
(1)	0	X	X	0	X	0
(7)	0	X	X	X	0	0
(8)	0	0	X	X	X	0
(5)	0	X	0	X	X	0
(-)	0	0	0	0	0	0

(m) - vertex number
(-) - missing vertex

Figure 4 - Example of memory mapping for a pair of boxes with common edge (1,7) of 3x3 grid graph of Figure 1.

common edge (1, 7). On a mesh-connected computer with p processors, $(p^{1/2}/(n_{h,k}+n_{h+1,k}+2))^2$ boxes can be operated on in parallel. The actual location of each box within the processor array must be precomputed for all levels of the separator tree before the recursive decomposition of A can start. Our box-compositon strategy greatly simplifies this problem since it allows the processor in which each coefficient is to be stored to be determined based simply on the connectivity of the graph.

Two additional ideas are essential to the successful use of the memory mapping scheme described above. First, it is helpful to think of the boxes as having an orientation: For even h, the separators are horizontal and the boxes lie along a north-south axis while for odd h the separators are vertical so that the boxes have an east-west orientation. Second, the merging of a pair of $a_{h,k}$'s (see above) is facilitated by dividing the vertices on the perimeters of the boxes into sequences that will contain different vertices depending on the position of a box in a pair.

Let $s_{h,k}$ be a separator at the h^{th} level in the separator tree. Then, for a North (East) box let sequence a (sqa) be the $(2n_{h-1,k}+n_{h,k}+1)$ first vertices and sequence b (sqb) the c_1, $s_{h,k}$(in reversed order) and c_0 vertices. Also, for a South (west) box let sequence a (sqa) be the $n_{h-1,k}$ first vertices, sequence b (sqb) the c_0, $s_{h,k}$ and c_1 vertices and sequence c (sqc) the remaining vertices. The steps involved in merging pairs of $a_{h,k}$'s can now be described as:

(i) For each north (east) box reverse the coefficients associated with the contact points about the data for the separators and move c_0 data to the processors containing this data for the south (west) box

(ii) For each north (east) box, reverse the order of the separator coefficients

(iii) Insert the data for sqa and sqb (minus the c_0 coefficients) for the north (east) box after the data for the sqa of the south (west) box: this automatically aligns the c_1 and $s_{h,k}$ coefficients for the two boxes

(iv) do matrix addition

The data movements of (i), (ii) and (iii) translate into shifts through the mesh-connected network. The important points are that all shift distances are given by the dimensions of the boxes and that the data for common edges automatically line up. Having merged the pairs of a_h's, the computation of $a_{h+1,k}$ becomes straightforward since all the required data is now stored in a local neighborhood of processors.

"Backsolving" and Performance Issues

PND reduces the problem of inverting (LDL^T decomposition) a large sparse matrix, A, to that of inverting a number of much smaller dense matrices by computing a recursive s(n)-factorization of A. Now, a choice has to be made as to which algorithms should be used for matrix inversion(or LDL^T decomposition) and multiplication on the MPP. We have implemented both systolic and non-systolic algorithms for these operations. In the case of inversion, a comparison was made between a non-systolic Gauss-Jordan and a systolic Givens rotation method (Ref. 1) This comparison showed that although the latter required much less inter-processor communications than the former, this was off-set by the MPP's broadcasting facility and the smaller number of arithmetic operations of Gauss-Jordan. A similar result was obtained for the matrix multiplication algorithms.

Finally, let us consider the "backsolving" part of PND. An inspection of equation (3) shows that in order to minimize the total amount of arithmetic required to solve a sparse, linear system using PND, some of the quantities computed during the factorization of A should be saved for the "backsolving" calculation. These include $x_{h,k}^{-1} y_{h,k}^{T}$ and $x_{h,k}^{-1}$. Since each processor of the MPP has only 1k-bits of local memory, it would be impossible to store quantities where they are computed. However, the processors also have access to 16k-bits of staging memory each. This gives the MPP an impressive total amount of storage and makes possible the saving of quantities described above. The overhead of moving data between local processor memories and the staging memory would be less than that incurred if quantities had to be recomputed during the "backsolving" stage.

The remaining issue is now how to perform the sparse multiplication $A^{-1}b$ given our memory mapping. There are two possible approaches - either to divide b into segments based on the vertices of the perimeters of a box or to store b as a single global data structure. We have chosen to implement the former strategy. Note that in principle, the sum-or tree of the MPP can be used to pipeline the solutions to many sparse linear systems, Ax=b, assuming A is the same for all of them. In particular, we can solve O(s(n)) systems with distinct vectors b in total time O(s(n)) using s(n)2 processors.

With the implementation of PND outlined in this section, and given the storage constraints of the MPP, estimates show that linear systems with up to 16k unknowns can be recursively factorized in about 40 seconds. The "backsolving" will take about 1 second. It should be noted that the size of the problems that can be tackled by our implementation of PND for mesh-connected parallel computers is only limited by available machine memory.

REFERENCES

1. Bojanczyk, A., R. P. Brent and H. T. Kung, "Numerically stable solution of dense systems of linear equations using mesh-connected processors," Technical Report, CMU-CS-81-119, Computer Sci. Dept., Carnegie-Mellon University, Pittsburgh, PA (1981).

2. Csanky, L., "Fast Parallel Matrix Inversion Algorithms," SIAM J. on Computing 5(4), 618-623 (1976).

3. George, J. A., "Nested Dissection of a Regular Finite Element Mesh," SIAM J. on Numerical Analysis 10(2), 345-367 (1973).

4. Hageman, L. and D. Young, "Applied Iterative Methods," Academic Press, New York (1981).

5. Leiserson, C. E., Jill P. Mesirov, Lena Nekludova, S. Omohundro and J. Reif, "Solving Sparse Systems of Linear Equations on the Connection Machine," Annual SIAM Conference, Boston, MA (1986).

6. Lipton, R., D. Rose and R. E. Tarjan, "Generalized Nested Dissection", SIAM J. on Appl. Math. 36, 177-189 (1979).

7. Pan, V., "Fast and Efficient Parallel Algorithms for the Exact Inversion of Integer Matrices," Technical Report, TR 85-2, Computer Sci. Dept., SUNYA, Albany, N. Y. (1985).

8. Pan, V. and J. Reif, "Efficient Parallel Solution of Linear Systems," Technical Report, TR-02-85, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA (1985).

9. Pan, V. and J. Reif, "Fast and Efficient Algorithms for Linear Programming and for the Linear Least Squares Problem," Technical Report TR-11-85, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA(1985).

10. Pan, V. and J. Reif, "Extension of the Parallel Nested Dissection Algorithm to Path Algebra Problems," Technical Report 85-9, Computer Sci. Dept., SUNYA, Albany, N. Y. (1985).

N87-26562

SOLUTION OF LARGE LINEAR SYSTEMS OF EQUATIONS ON THE MASSIVELY PARALLEL PROCESSOR

Nathan Ida Kapila Udawatta
Electrical Engineering Department
The University of Akron
Akron, Ohio 44325

ABSTRACT

The Massively Parallel Processor, has been designed as a special machine for specific applications in image processing. As a parallel machine, with a large number of processor that can be reconfigured in different combinations it is also applicable to other problems that require a large number of processors. This work investigates the solution of linear systems of equations on the MPP. The solution times achieved are compared to those obtained with a serial machine and the performance of the MPP is discussed.

INTRODUCTION

The advantage of a parallel computer is in its potential ability to solve large problems in realistic solution times. In particular, as the improvements in speed of single processor computers approach intrinsic limits, the appeal of parallel processing becomes more significant. Yet, beyond the fact that some problems have little or no natural parallelism, the performance of such machines is not known with any accuracy. The ideal performance is of course well defined. It depends on the number of processors and their speed. Thus, upper and lower limits on computation speed can always be obtained. This does not take into account a variety of considerations like I/O and other degradation factors. Thus, in the ideal case, the speedup achieved through parallel processing is equal to the number of processors [1]. A variety of factors influence the performance to reduce the speedup considerably. Among these factors, the competition of processors for hardware and the interaction between the parallel processes are the most important [2]. Obviously, the algorithm to be executed has a drastic influence on the performance. Ideally, the algorithm has intrinsic parallelism such that there is no need to idle processors. In reality, this is not the case and there are always serial operations to be performed. Again, in the limit, when no parallel operation can be performed, the parallel processor is used as a serial computer.

Estimating the performance of a parallel processor for any particular type of problems is not a trivial process. In many cases this cannot be done without actually solving the problem and then evaluating the performance. Thus, the need to evaluate a computer's performance on well documented problems or benchmarks becomes extremely important, particularly with parallel processors. With all this, one can safely assume that some degree of parallelism does exist in most algorithms and therefore an improvement in solution time compared with serial machines can be realized.

The work presented here attempts to establish the usefulness of a parallel processor, the Massively Parallel Processor (MPP), for the solution of systems of linear equations. No attempt is made to either optimize code or to establish exact performance figures. Such an attempt is beyond the scope of this paper and would require considerably more work. Instead, the emphasis is on a particular simple algorithm and on the comparison of performance with a serial computer (the Microvax II). The type of systems solved are those arising from the application of the finite element method to engineering applications. The finite element method is particularly computationally intensive. By using a parallel processor, it is conceivable that considerably faster solution times can be achieved or, alternatively, larger problems can be solved.

The solution method chosen for evaluation is the Gauss elimination method. It is used as representative to direct solution methods and because for the majority of practical applications it is used almost exclusively. The results will therefore be useful for implementation of other similar methods like the Gauss-Jordan or the Choleski decomposition methods.

GAUSS ELIMINATION

Consider the following system of n equations with n unknowns:

$$a_{11}X_1+a_{12}X_2+a_{13}X_3+\dots\dots+a_{1n}X_n=C_1 \quad (1a)$$

$$a_{21}X_1+a_{22}X_2+a_{23}X_3+\dots\dots+a_{2n}X_n=C_2 \quad (1b)$$

$$a_{31}X_1+a_{32}X_2+a_{33}X_3+\dots\dots+a_{3n}X_n=C_3 \quad (1c)$$

$$\vdots$$

$$a_{n1}X_1+a_{n2}X_2+a_{n3}X_3+\dots\dots+a_{nn}X_n=C_n \quad (1n)$$

The system in Eq. (1) is assumed to be symmetric, positive definite for the purpose of this work although none of these requirements is necessary in general. The elimination is done in the following order:

Equation 1a is divided by its diagonal coefficient to obtain

$$X_1+a_{12}/a_{11}X_2+a_{13}/a_{11}X_3+\dots \\ \dots\dots+a_{1n}/a_{11}X_n=C_1/a_{11} \quad (2)$$

Equation (2) is multiplied by the first coefficient of (1b)

$$a_{21}X_1+a_{21}a_{12}/a_{11}X_2+\dots\dots \\ \dots+a_{21}a_{2n}/a_{11}X_n=C_1a_{21}/a_{11} \quad (3)$$

Eq. (3) is subtracted from Eq. (1b) to eliminate the coefficient of X_1 from Eq. (1b). In the next step, Eq. (2) is multiplied by the first coefficient in Eq. (1c). Subtraction as previously results in elimination of the coefficient of X_1 in Eq. (1c). Repeating this for all the remaining coefficients in the first column results in the following system:

$$a_{11}X_1+a_{12}X_2+a_{13}X_3+\dots\dots+a_{1n}X_n \\ =C_1 \quad (4a)$$

$$a'_{22}X_2+a'_{23}X_3+\dots\dots+a'_{2n}X_n \\ =C'_2 \quad (4b)$$

$$a'_{32}X_2+a'_{33}X_3+\dots\dots+a'_{3n}X_n \\ =C'_3 \quad (4c)$$

$$\vdots$$

$$a'_{n2}X_2+a'_{n3}X_3+\dots\dots+a'_{nn}X_n \\ =C'_n \quad (4n)$$

In this set, all coefficients were altered except for those in the first row. It is important to note that the operations are done on a whole row at a time, a property which will be exploited later for parallel calculation.

The elimination proceeds by repeating the whole process starting with the second equation. After n-1 such elimination steps, the original system is reduced to an upper triangular system:

$$a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + \cdots\cdots + a_{1n}X_n = C_1 \quad (5a)$$

$$a'_{22}X_2 + a'_{23}X_3 + \cdots\cdots + a'_{2n}X_n = C'_2 \quad (5b)$$

$$a''_{33}X_3 + \cdots\cdots + a''_{3n}X_n = C'_3 \quad (5c)$$

$$\vdots$$

$$a^n_{nn}X_n = C^n_n \quad (5n)$$

The elimination step, which is done in place (i.e. in the same locations the original matrix resides), is followed by a backsubstitution step. This starts by calculating X_n

$$X_n = C^n_n / a^n_{nn} \qquad (6a)$$

The rest of the unknowns are calculated as:

$$X_i = [C_i - \sum_{j=i+1}^{n} a_{ij}X_j]/a_{ii} \quad (6b)$$

where $i = n-1, n-2, \ldots\ldots, 1$ and C_i are the modified right hand sides in Eq. (5).

THE MASSIVELY PARALLEL PROCESSOR

For the solution of linear systems, the two most important aspects related to the MPP are the number of memory planes in the ARU and the size of the staging memory available for use. Although the ARU contains 1024 bit planes of memory, the programmer can use only bit planes from 0 to 973. Bit planes from 974 to 1023 are reserved for use by system software (Control and Debug). This limits the number of 128*128 real arrays (32 bit floating point arrays) in the ARU to 30. Without taking into consideration any necessary scratch arrays, the capacity of the ARU is limited to one 640*640 real array or an equivalent size array. In practical terms, since scratch arrays are needed, the limit is lower. A matrix of 512*512 is the limit if increments in size of 128*128 are to be used.

The staging memory has a capacity of 32 Megabytes. This limits the number of stored 128*128 real arrays in the stager to 512.

It is important to note that the parallel Pascal callable I/O procedures can transfer only one 128*128 array in or out of the ARU at any one time. This makes it necessary for any array larger than 128*128 to be blocked into sub-arrays of 128*128 by assigning the larger array two more dimensions. Blocking of a 512*512 array is given in Fig. 1.

Other aspects of programming on the MPP are not discussed here although they are necessary for implementation. These can be found in a variety of references [3,4,9-15].

IMPLEMENTATION ON THE MPP

Since the minimum data array that can reside in the ARU is 128*128, as a first step the Gaussian elimination was implemented for an array of this size. In addition to the data array, three more real arrays (32 bit) and one boolean array were used as scratch space for the Gaussian elimination. That is, 97 bit planes were used as scratch space.

The algorithm used for Gaussian elimination of an 128*128 array can be written in pseudo code as

```
DO UNTIL # ROWS = 128
BEGIN
NEW  ARRAY <-- PIVOT ROW FROM  ORIGINAL
               ARRAY;
PIVOT ELEMENT ARRAY <-- NEW ARRAY;
MULTIPLIER ARRAY <-- ORIGINAL ARRAY;
NEW  ARRAY <-- NEW ARRAY/PIVOT  ELEMENT
               ARRAY;
NEW   ARRAY  <-- NEW   ARRAY*MULTIPLIER
                 ARRAY;
ORIGINAL ARRAY <-- ORIGINAL ARRAY - NEW
                   ARRAY;
END;
```

The parallel Pascal code for Gaussian elimination of an 128*128 array was used as the building block and was extended for larger arrays. An array of 512*512 was chosen since this is the maximum array that can reside in the ARU. Scanning of the 512*512 array, by a block of 128*128, is given in Fig. 2.

A theoretical calculation of timing for an array of 1024*1024 was also done. The 1024*1024 array was blocked into four sub-arrays of 512*512 and at any given time one 512*512 sub-array resided in the ARU. The actual time needed for loading/unloading the ARU with a 512*512 real array is 590 ms. The time needed to eliminate 2 columns from a 512*512 real array is 10 ms. A real array of 512*512 occupies 512 bit planes. This leaves the ARU with 14 real arrays to be used as scratch space. Of these, one plane is necessary to use as a boolean array for the mask plane. This makes it possible to save 14 "coefficient" arrays in the ARU. By saving 14 coefficient arrays it is possible to eliminate 2 columns from any of the four 512*512 sub-arrays. the Gaussian elimination on the 1024*1024 real array can be written in pseudo code as

```
DO L = 1 To 2
 BEGIN
  DO K = 1 TO 256
   BEGIN
    DO I = L TO 2
     DO J = L TO 2
      BEGIN
       IF L = 1 THEN
        BEGIN
         ELIMINATE 2 COLUMNS FROM
         SUB-ARRAY;
          UNLOAD SUB-ARRAY TO STAGER
          WHILE LOADING NEW SUB-ARRAY
          TO ARU;
         END;
        ELIMINATE 512 COLUMNS FROM
        LAST SUB-ARRAY;
       END;
      END;
     END;
```

RESULTS

The solution times for Gaussian elimination of an 128*128, 512*512 and 1024*1024 real arrays on the MPP are summarized and compared with the solution times for the same arrays on a Microvax-II. The performance of the MPP compared to a serial computer is illustrated in Fig. 3.

Arrays below the size of 27*27 can be solved on the Microvax-II computer and obtain the same performance as on the MPP. The array size below which it is more economical to use a serial computer, depends on the type of the serial computer.

But, as seen in Table 1 and Fig. 3, the performance of the parallel computer (MPP) improves with the increase of the size of the data array and drops sharply once the data array exceeds the memory space of the ARU. Taking into account the actual times involved, the MPP is much faster than a serial machine.

It is important to note that the algorithms used for the MPP were not optimized, since the study involved only understanding the performance of the MPP in solving large systems of linear equations and not the precise evaluation of the MPP performance. Fig. 3 clearly shows the tendency of the behavior of the MPP. In addition to the algorithms being nonoptimized, no attempt was made to taylor the algorithm to a particular type of problem. For example, consider the matrices generated by finite element modeling which are banded (and, in many cases, symmetric).

For this type of matrices great advantage can be taken of the fact that the matrix has a limited bandwidth. The shaded areas in Fig. 4 show the sub-arrays (128*128) that could be used while solving (scanning) a 512*512 banded matrix with a bandwidth which is less than 128.

The performance can also be changed by using different blocking techniques for larger arrays. It will be interesting to see the performance when a 1024*1024 array is blocked not as four arrays of 512*512, but four 1024*256 arrays. In this way, the optimal blocking of large arrays for particular problems can be chosen.

The experience gained here clearly shows not only that the solution of large systems of equations is possible and faster than that possible on serial machines but also that there is an alternative between using large arrays or smaller arrays with large memory. The structure of the MPP allows the user to fit a matrix as large as the memory of the stager. Clearly, if the matrix can be fitted in the ARU, the solution will be faster than for the case where parts of the matrix need to be retrieved from the stager or from the front end computer.

From the results presented above it can be seen that an 1024*1024 array (for example) will solve a 1024*1024 matrix in about 48 ms which means a speedup of about 69,320 against the speedup of 6.6 for an 128*128 array.

At the same time, an 128*128 array with large memory (large enough to contain the matrix) can solve a 1024*1024 matrix in 4.21 s, resulting in a speedup of 823 compared to 6.6 with a 1k memory.

As it stands now, the tendency seems to be towards larger arrays but it will be interesting to study the performance of these machines with larger memories which are by nature less expensive than processors.

CONCLUSIONS

The results presented in this work show that the Massively Parallel Processor is particularly suited to the solution of problems which can fit in the ARU. The speedup obtained compared to serial computers is in the hundreds. Not surprisingly, the performance deteriorates somewhat when parts of the matrix need to be brought in from the stager. The deterioration in performance is quite dramatic when the matrix does not fit into the stager and there is a need to perform considerable

I/O. With all this, the solution is always faster than for serial computers.

REFERENCES

1. K. Hwang, F. A. Briggs, "Computer architecture and parallel processing", McGraw-Hill Book Company, New York, 1964.

2. T. Axerold, "Comparing the performance of parallel computers", Compcon, spring 1985.

3. K. E. Batcher, "Design of a massively parallel processor", IEEE Transactions on Computers, Vol. C-29, No. 9, September 1980.

4. K. E. Batcher, "Architecture of the MPP", IEEE Computer Society on Computer Architecture for Pattern Analysis and Image Database Management Proceedings, October 1983, pp. 170-174.

5. P. B. Hansen, "The programming language concurent Pascal", IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975.

6. "MPP I/O control unit", Goodyear Aerospace Corporation, GER-16679, April 1983.

7. "MPP main control unit", Goodyear Aerospace Corporation, GER-16659, April 1983.

8. "MPP PE control unit", Goodyear Aerospace Corporation, GER-16650, June 1983.

9. "MPP Staging memory", Goodyear Aerospace Corporation, GER-16964, March 1981.

10. "MPP staging memory manager", Goodyear Aerospace Corporation, GER-17062, April 1983.

11. "MPP user's guide", NASA Goddard Space Flight Center, January 1986.

12. "Parallel Pascal callable I/O procedure", NASA Goddard Space Flight Center, January 1986.

13. "Parallel Pascal language reference manual", MUD-210, Version 2.

14. A. P. Reeves, "Parallel Pascal: an extended Pascal for parallel computers", Journal of Parallel and Distributed Computing 1, Vol. 64, 1980.

15. "Theory of the MPP hardware operation", Goodyear Aerospace Corporation, GER-17143, April 1983.

Table 1. Processing speeds of Microvax-II and MPP

Array	Microvax	MPP	Speedup
16*16	10 ms	48.07 ms	0.2
27*27	50 ms	48.07 ms	1.0
128*128	5500 ms	48.07 ms	114
512*512	340 sec	1.272 sec	267
1024*1024	57.76 min	8.12 min	6.6

ORIGINAL PAGE IS
OF POOR QUALITY

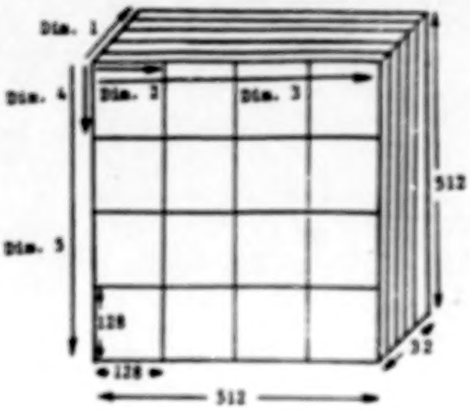

Dim. 1
Dim. 4
Dim. 2
Dim. 3
Dim. 5
512
128
128
512
32

Figure 1. Blocking and dimensioning of 512*512 array into sub-arrays 128*128 insize.

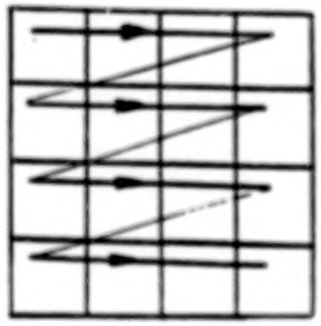


Figure 2. Mode of scanning a 512*512 array in blocks of 128*128.

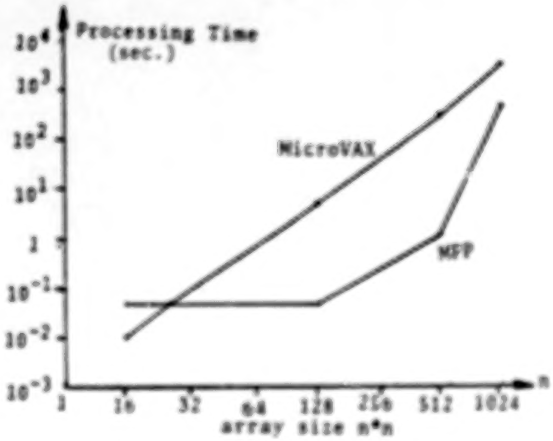


Figure 3. Processing time of the MPP and the Microvax-II vs. array size.

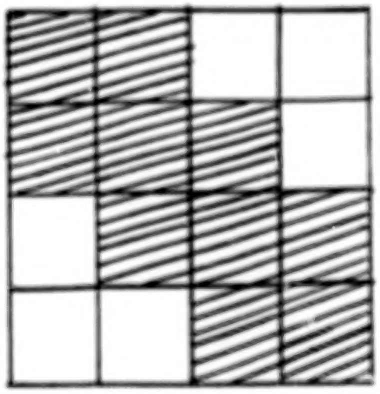


Figure 4. Possible scanning of a banded matrix with bandwidth of 128 or less.

N87-26563

The Factorization of Large Composite Numbers on the MPP

by

Kathy J. McKurdy, Goodyear Aerospace Corporation
Marvin C. Wunderlich, Department of Defense

1. Introduction: The continued fraction method for factoring large integers (hereafter referred to as CFRAC) was an ideal algorithm to be implemented on a massively parallel computer such as the MPP. The history of this effort goes back many years. The first effort to implement this algorithm on the ILLIAC IV was thwarted first by an inadequate resolve on the part of a funding agency and then by the sudden dismantling of the computer itself. The second attempt was to put the program on the English DAP with the able assistence of Dennis Parkinson of Queen Mary College, London. This effort was spoiled by the inadequate amount of time the second author was able to spend in England in the summer of 1982. He was finally able to devote full time on the NASA MPP implementation in the summer of 1984 and by September of 1985, the authors suceeded to factor their first 60 digit number on the MPP using about 6¼ hours of array time. Although this result added about 10 digits to the size number we could factor using CFRAC on a serial machine, it was already badly beaten by the implementation of Jim Davis and Diane Holdridge on the CRAY-1 using the quadratic sieve, an algorithm which is clearly superior to CFRAC for larger numbers. This work does illustrate, however, an algorithm which is ideally suited to the SIMD massively parallel architecture and we describe some of the modifications which were needed in order to make the parallel implementation effective and efficient.

2. The Continued Fraction Algorithm. To describe this method, we must first describe a method for generating small quadratic residues, mod N, where N is the composite number we wish to factor. An integer Q is said to be a quadratic residue, mod N, if an integer A exists such that

$$Q \equiv A^2 (mod\, N). \tag{1}$$

Pairs (Q,A) satisfying (1) can be generated by expanding the simple continued fraction of $\sqrt{N}$. Space and time prevents us from elaborating on this subject so it must suffice to simply describe the algorithm. If we initiate the variables $d = [\sqrt{N}]$, $A_{-1} = 1$, $P_0 = 0$, $Q_0 = 1$, $A_0 = d$, we can generate the pair (Q_{k+1}, A_k) recursively from earlier pairs by the formulas

$$q_k = [(P_k + d)/Q_k] \tag{2a}$$

$$P_{k+1} = q_k Q_k - P_k \tag{2b}$$

$$Q_{k+1} = (N - P_{k+1}^2)/Q_k \tag{2c}$$

and

$$A_{k+1} \equiv q_k A_k + A_{k-1} (mod\, N). \tag{2d}$$

Then it can be proved that

$$A_k^2 \equiv (-1)^{k+1} Q_{k+1} (mod\, N) \tag{3}$$

and

$$Q_k \leq 2\sqrt{N}. \tag{4}$$

Now the clever reader may recognize a novel factoring method here. If N is the composite number to be factored, simply generate the pairs (Q_{k+1}, A_k) until Q_{k+1} is itself a square and k is even. Then if $Q_{k+1} = X^2$, we have $(A_k)^2 \equiv X^2$ (mod N) or

$$N \mid A_k^2 - X^2 = (A_k - X)(A_k + X). \tag{5}$$

If N = pq where p and q are both primes, there is an even chance that one prime will divide $A_k - X$ and the other will divide $A_k + X$ and in this event, computing the greatest common divisor $GCD(N, A_k - X)$ will reveal either p or q and N is factored. If this doesn't happen for Q_k, keep generating (Q_{k+1}, A_k) pairs until a square Q_{k+1} works. From (4), there are about $2\sqrt{N}$ different possible values of Q_k and among them there will be about $N^{.25}$ squares. Thus, we should have to generate about $N^{.25}$ values of Q_k before N is factored. The clever reader should

PRECEDING PAGE BLANK NOT FILMED

N87-26563

congratulate himself for noticing this since his/her algorithm is already vastly superior to the simple divide and factor method which requires min(p,q) operations. However, this still consumes too much computer time to be really competitive with the leading state-of-the-art methods.

What we do in CFRAC is to obtain collections of Q's whose product is a square. Suppose I is a set of indices which defines such a collection. We deduce from (3) that

$$X^2 = \prod_{i \in I} (-1)^i Q_i \equiv \prod_{i \in I} A_{i-1}^2 = Y^2 (mod\, N) \tag{6}$$

and since $N = pq \mid X^2 - Y^2 = (X-Y)(X+Y)$, a factor can be produced by computing GCD(X − Y,N). To find a collection of Q's whose product is a square, we attempt to factor each Q over a fixed set of primes $p_1, p_2, \dots, p_k$ and represent each Q_i which factors completely with a binary vector $(\varepsilon_0, \varepsilon_1, \varepsilon_2, \dots, \varepsilon_k)$ where ε_0 is ±1 according to which of $\pm Q_{i+1} \equiv A_i^2$ in (3) and ε_j, $i < 0$, is one if p_j divides Q_i to an odd power and zero otherwise. Note that the vector ε_i is the zero vector if and only if the corresponding value Q_i is a square and a quadratic residue, mod N. When we have factored more than k of these values Q, we can form a matrix M with these vectors and M will have more rows than columns. We can perform a Gaussian reduction on this matrix, produce zero rows and each such zero row will represent a collection of Q's whose product is a square. Thus (6) is satisfied and we may be able to factor N by computing the appropriate GCD. If this doesn't work, we can use another such collection of Q's since a collection whose product is a square is generated with each zero row produced from the Gaussian reduction.

3. The MPP Implementation: The most time consuming aspect of this algorithm is the factorization of the Q's. Each value of Q must be divided by each of the k primes in the base of primes until the number of factored Q's exceeds k. To factor a typical 60 digit number, one must attempt a factorization of over 100,000,000 values of Q using a base of 4,000 prime numbers and this requires 4×10^{11} division instructions. The MPP implementation described in this section performs this task very efficiently as well as the task of generating the (Q,A) pairs and the final Gaussian reduction. We shall discuss these three parts of the implementation separately.

3.1. Stepping. The generation of the (Q,A) pairs is clearly a recursive process and there is no obvious way to employ 16,384 processors to accomplish this task in parallel. However, Daniel Shanks and Hugh Williams have devised a very clever algorithm for taking giant steps in the recursion process. Knowing (Q_{s+1}, A_s) and (Q_{t+1}, A_t), one can generate a term (Q_{u+1}, A_u) whose u is very near s + t. The time needed for this composition is a constant which does not depend on s and t. This enables us to generate a pair (Q_{t+1}, A_t) where t is as large as we please by composing a succession of terms with a nearby term approximately $\log_2 t$ times. For this particular implementation, we first generated (Q_{t+1}, A_t) for t near one million (1M) and then generated 16,384 pairs (Q_{r+1}, A_r) where r was 1M, 2M, ..., 16384M. This was done on a fast serial machine. Then we put a pair (Q_{r+1}, A_r) in each of the 16,384 processors and generated successive terms in parallel using the recursion (2). Since the terms are 1M apart, we can generate as many as 16,384,000,000 terms before there is any danger of the same pair being generated in neighboring processors.

A serious problem arose in trying to implement the recursion described in (2) in parallel on the the MPP. The numbers involved are quite large. For a 60 digit factorization, the A's are 200 bits, the P's and Q's are each 100 bits and it was not possible to perform all the necessary arithmetic in the 900 bits of available memory in the ARU. For this reason, we used a fast bit plane I/O system developed by Goodyear Aerospace to use the staging memory as auxiliary storage. Using that package, storage can be allocated in the stager memory in the same way that storage is allocated in the ARU. A set of SEND macros exists which moves data between the stager and the ARU. SEND macros also exist to move data between the ARU and the Host, and the MCU and host. This package has essentially doubled the available memory for doing computational processing and has also provided an easy-to-use I/O management package for the entire algorithm. Data moves between the ARU and the stager can concur with computational operations which considerably reduces the extra time needed for the data swapping.

3.2. The Factoring. Having a different value of Q in each processor and the corresponding value A in the staging memory, the program now proceeds to attempt a factorization of the Q's over a set of primes stored as scalars in the MCU. Actually, the MCU only contains the differences between the consecutive primes. It also should be pointed out that the prime base consists of the smallest 4000 primes which are possible divisors of the Q's, and since the Q's are quadratic residues, mod N, only primes p for which N is a quadratic residue, mod p, are possible divisors of Q and so the prime base consists of the smallest 4000 primes having this property.

The fundamental operation is to divide all the values of Q by the integer p in one simultaneous ARU instruction and flag those P.E.'s where the remainder is zero. This operation is then repeated in the flagged processors, toggling a parity plane until all processors produce a non-zero remainder. The parity plane will contain a 1 or 0 indicating whether the primes p divided Q to an odd or even power. The difficulty with this method is that the divide instruction must be repeated t times where t is the largest power of p which divides any of the 16,384 values of Q. This can be rather large for small primes p. Certainly the divide instruction must be executed at least twice for each batch of Q's so the efficiency of the algorithm will be at most .5.

A two step algorithm is employed which avoides this difficulty. In the first step, the values Q are divided by each p in the prime base exactly once, and a table is collected in each PE which contains the set of primes p_{k_1}, p_{k_2}, ... , P_{k_l} which divides the Q in that particular processor. This table contains between 12 and 15 primes for each Q. Then the single step method described above is applied to the primes in the table in order to ascertain the parity of the power of p which exactly divides Q. This way the inefficiency of the single step procedure only affects about 15 division instructions rather than 4000. A serious difficulty arises, however, when attempting to implement this two step procedure. In the first step, we will be dividing 16,384 Q's by a scalar prime p setting a FLAG to 1 wherever the remainder is zero. Then in all processors in which FLAG = 1, the prime p must be put at the end of a short table in the ARU. However, the address of the end of the table is different in each processor. The "lock step" SIMD character of the MPP does not permit storing a value in different locations in different PE's.

We are indebted to Kenneth Batcher of Goodyear Aerospace for providing an ingenious solution to this problem. We begin by dividing the first 200 primes into the 16,384 Q's and setting 200 bit planes to flag the values Q which were evenly divisible by the primes. We now go back through the 200 bit planes and using them as flags, push onto the shift register the least significant 2 bits of the primes that evenly divide the Q's. At this point, the shift register in each PE contains the table p_{k_1}(mod 4), p_{k_2}(mod 4), ... p_{k_l}(mod 4) where the primes p_{k_i} are those which exactly divide the Q in that processor. The shift register is now stored in the least significant 2 bits of each entry of the table we are attempting to construct. Then the same procedure is followed for the 3rd and 4th significant bits, the 5th and 6th bits and so on until the 19th and 20th significant bits of the primes. No prime in the factor base ever exceeds 20 bits. The total number of bit instructions used in this complicated procedure is the same as if there were a variable address store in the SIMD instruction set. This routine was coded by K. Batcher in PEARL on the PECU. The value 200 was chosen because this number of bits was the most we could spare in the 1000 bit ARU. This procedure would be much easier to program on an MPP with larger memory.

This entire 3-step algorithm was executed by an MPP program called FACTOR and was able to do one complete batch of 16,384 values of Q in about one second of MPP time. Since the minimum number of division instructions needed to accomplish this task is 16,384 x 4000 = 65,536,000, this parallel routine operates at an average rate of 15.25 nanoseconds per instruction. The central instruction used in the program divided a 20 bit prime into a 100 bit Q and this used an optimally coded PEARL instruction which uses about two thousand 100 nanosecond cycles and this averages out to 12.21 nanoseconds per division instruction. From this, it follows that the FACTOR program operated at an efficiency rate of 12.21/15.25 or about 80%.

The program to generate the next batch of Q's and A's took considerably longer than was anticipated, mainly because of the tiny amount of core allocated to each P.E. Each STEP took about .1 seconds, but since it took so much less time than FACTOR, there was no driving need to optimize this procedure.

3.3. The Gaussian Reduction. There was no need to optimize this part of the program either. For the earlier factorizations, a VAX program was used to perform the reduction and it took about 25 minutes of VAX time. By way of comparison, over 6 hours of time was used to perform the factoring on the MPP. On the other hand, the very existence of 16,000 x 1,000 = 4,000 x 4000 bits of readily accessible memory made the development of an MPP-based Gaussian elimination program an irresistible temptation. For this purpose, it would have been preferable to have 4000 PE's each having 4000 bits of memory. Then, we could store each row of the matrix in one bit plane of the MPP. In this situation, we had to store one row of the matrix in a quarter of a bit plane and this was done by an arrangement of stripes in which columns 0, 4, 8, . . . , 124 of a 128 x 128 bit plane represented one row of the 0-1 matrix M. With this arrangement, an entire 4000 x 4000 matrix of bits can be stored in the entire MPP array memory, leaving very little memory for anything else.

The usual procedure to do a Gaussian reduction on a bit matrix is to do a series of elementary row operations until a matrix is obtained having a single 1-bit in each row and column -- i.e, a permutation of the identity matrix. One also performs the same elementary row operations on a history matrix which was set at the beginning to the identity matrix. If, at anytime in this process, a zero row is produced, the ones in the history matrix identify the rows on the original matrix which were initially linearly dependent. Of course, having the entire memory of the MPP used to store the original matrix M, there is no room for a history matrix. Therefore, we utilized an in-place algorithm first suggested to the author by Dennis Parkinson and completely described in [2]. We shall not give a detailed description of the procedure in this paper, but the idea is to use the "zero-space" produced by elementary row operations to store the "one-space" generated in the history matrix.

Theoretically, one should be able to always reduce a n by n bit matrix in $1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2$ elementary row operations and if one row operation takes one MPP cycle of 100 nanoseconds and $n = 4000$, the time should be just under one second. Of course, the tight loop requires branch instructions and tests which themselves require at least 100 nanoseconds apiece. When the reduction program GELIM finally worked, a stop-watch timing of the program showed that usually 8 seconds were required to reduce the sparse matrix generated by CFRAC.

The operation of this program highlighted an interesting but disturbing feature of massively parallel processing. The factoring and Q generation portions of the program were completely fault tolerant. Every time a plane of (Q, A) pairs were generated, the relationship $\pm Q \equiv A^2 \pmod{N}$ was tested and those processors which failed the test were disabled with a mask for the rest of the run. On one occasion, there were 7 or 8 processors disabled after several hours of MPP computation. In GELIM, however, no such tolerance was permitted. If just one bit were at fault anywhere in the execution of the algorithm, the data obtained by GELIM was rendered completely useless. This actually occurred when the first factorization of a never-before-factored number was attempted. In August of 1985, The second author was scheduled to deliver a talk at a computational number theory conference in Arcata, California, and on the morning of the talk, GELIM was to triumphantly produce a set of linearly dependent rows of the matrix M. However, when the number of one bits in each column of the dependent set was counted, about 3% of the 4000 columns was odd, not even as required. Apparently, a single bit of the matrix was in error somewhere in the algorithm and by the end of the run, the fault spread to infect about 6% of the column data. He never gave that talk but rather rushed back to Goddard where he quickly put together a hasty GELIM on the VAX which produced the desired factors within 2 weeks of the aborted talk. It wasn't until last February when the first author found the hardware bug in the MCU which caused the occasional error and patched the MPP program so that it would run correctly each time.

Perhaps error correction is really needed in the ARU memory chip.

4. The Large Prime Variation. This is an improvement of the basic factoring strategy which has been utilized in all implementations of CFRAC. If, after a quadratic residue has been divided by all the admissible primes which are less than a number x, the remaining unfactored part F is less than x^2, then F itself must be a prime. Since F is not in the factor base, we call these large prime factorizations. If two different quadratic residues, Q_1 and Q_2, have large primes factorizations with the same large prime F, then the product $Q_1 * Q_2$ will have a factorization of the form

$$Q_1 Q_2 = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_j^{\alpha_j} F^2 .$$

where all the p_i are in the factor base. In the factoring process, this means that when Q_i and Q_j have the same large prime factorization and produce the 0-1 vectors ε_1 and ε_2, the exclusive OR of ε_1 and ε_2 can be added to the matrix M. In practice, very few pairs of large prime factorizations have the same large prime F when F is substantially larger than the largest prime in the factor base $p_k = x$. In the MPP implementation of this variation, all the large prime factorizations with $F < x^2/10$ were saved using a binary tree contained in the host VAX and whenever a collision occured between two Q's, the exclusive OR of the two variables was added to the matrix M. This variation was added to the MPP factoring program by the first author and despite all the additional overhead involved with the procedure, it improved the performance by nearly 1.5 for numbers in the 60 digit range. It is generally believed that the usefulness of this variation will diminish as the size of the number increases.

5. Results. The table below summarizes the results of five factorizations of four different large numbers. The first column defines the origin of the number which was factored. In every case, the expression in column 1 had some algebraic and small known factors which were divided out of the number before they were processed by the MPP. Column 2 indicates the size in decimal digits of the number after these smaller factors were divided out. Column 3 indicates whether or not a straight CFRAC was employed or the large prime variation, CFRAC-LP. Columns 4 and 5 indicate the computer time used. The large amount of VAX time needed for the first two factorizations was due to an inefficient proceedure for computing the product of the Q's. Columns 6 and 7 lists the total number of Q for which a factorization was attempted and the rate at which the Q's were processed. Note that the large prime variation reduced substantially the number of Q's needed for the total factorization.

References

1. M. A. Morrison and J. Brillhart. "A Method of Factoring and the factorization of F_7", *Math Comp.*,29(1975), 183 - 205

2. D. Parkinson and M. C. Wunderlich, "A compact algorithm for Gaussian elimination over GF(2) implemented on highly parallel computers", *Parallel Computing* , 1 (1984).

3. M. C. Wunderlich, "Implementing the continued fraction factoring algorithm on parallel machines", *Math. Comp.* 44(1985) 251-260.

FIVE FACTORIZATIONS ON THE MPP

Number	Digits	Method	MPP hours	Total hours	Q attempted	Qs/MPP sec
$2^{299}-1$	60	CFRAC	6.4	10.9	309,657,600	13,440
$2^{405}-1$	60	CFRAC	4.0	11.5	182,894,592	12,701
$5^{171}+1$	62	CFRAC	14.0	14.5	646,791,168	12,833
$2^{405}-1$	60	CFRAC-LP	2.8	3.4	93,028,352	9,120
$5^{149}+1$	64	CFRAC-LP	9.75	10.4	395,986,512	11,281

N87-26564

A GENERIC APPLICATIONS SUBROUTINE LIBRARY FOR THE MPP

Michael L. Gough, W. David Wildenhain
Science Applications Research
Lanham, MD

ABSTRACT

A new methodology to increase the utility of the MPP has been developed, and will be presented here as an addition to the current methods of using the MPP. This methodology provides for the development of an MPP-side abstraction layer that is callable from any host-side high level language. Routines in the abstraction layer have the option of using a powerful software tool for accessing the stager as "virtual memory". An additional abstraction layer that allows for remote access to the MPP via DECnet will be discussed. This integrated approach to programming the MPP is a valuable tool for the implementation of interactive user driven systems that require the computational capabilities of the MPP as well as a controlled "user view". It is expected that this methodology will be used to integrate the MPP into many such systems, and thus promote greater use of the MPP by scientific researchers who are accustomed to user friendly environments.

Keywords: Software Design, Virtual Memory, Network Communications, Interactive Systems

BACKGROUND

Although the architecture of the MPP is a departure from the standard Von Neumann architecture, there is no algorithm that can be implemented on the MPP that cannot be implemented on a Von Neumann machine. The only benefit of using the MPP for scientific applications is the dramatic increase in execution speed that is gained for some algorithms. Much research has gone into transforming algorithms so that they can take advantage of the MPP's parallel architecture. With the speed increase that is attainable through the use of the MPP, it is possible to perform heavy computational tasks interactively. It is clear that the scientific community could greatly benefit from such interactive computing power.

This effort represents the first use of the MPP as part of a user friendly system which allows a researcher to activate tasks on the MPP transparently. By using the MPP in this manner, it is possible to provide the researcher with a set of generic software tools which perform large computational tasks within a user friendly interactive environment. The MPP can thus provide a valuable service to the large established group of users who are accustomed to using pre-packaged systems for scientific investigation.

The motivation for the work which led to this methodology was the desire to use the MPP as part of an interactive system which produces animated graphics on a specialized graphics device. This project involved the development of a generic software package that would transform any scientific data set into a uniform quadrilateral mesh. It became apparent that it was impractical to implement such an algorithm on the VAX, and that the MPP was well suited for this algorithm, but it was not clear that the MPP could be integrated into a large operational interactive system.

ENVIRONMENT

The MPP consists of a 128 x 128 array of microprocessors, each of which is equipped with 1/8k bytes of memory, and an MCU (Main Control Unit) which commands the processors in the array. The MCU has access to 64k of memory, which is used for executable code, as well as data. An auxiliary memory called the stager provides an additional 2k per processor in the array. Early in this investigation it was decided that the algorithms to be implemented on the MPP could not be limited to the 1/8k memory per processor and that the stager would be used, despite the obvious performance cost.

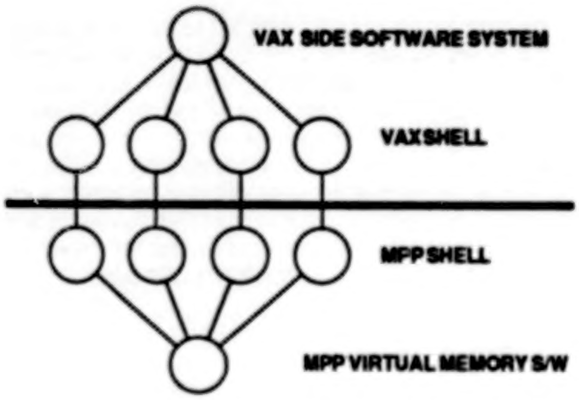
The front end machine for the MPP is a VAX 11/780. The VAX is well suited for user interaction, network access, and I/O to a wide variety of devices. It can also support large software systems that can not be ported to the MPP. Typical MPP applications are loaded into the MCU by the VAX, and activated. Control is not returned to the VAX until the completion of the application running in the MCU. In order to support truly interactive systems, a mechanism had to be devised to invoke various MPP primitives and return control to the VAX for further user interaction. At the discretion of the user, the MPP would be re-activated to perform another task. The overhead for MCU activation is 40 ms. This is negligible when used to return control to the host machine for user interaction.

PRECEDING PAGE BLANK NOT FILMED

MPP SHELL

The MPP Shell is a subroutine library of VAX callable routines which activate Parallel Pascal subroutines that execute in the MPP. An MPP Shell subroutine called MPP_open is called to allocate the MPP, and perform required buffer initialization. A subroutine called MPP_close is used to perform all necessary termination, including the deallocation of the MPP. The MPP_load routine is used to load data into the array memory, and the MPP_unload routine is used to unload data from the array to the host memory. The remainder of the MPP Shell subroutines perform computational tasks.

Each Parallel Pascal subroutine is called via a host side header subroutine as pictured below, which issues a call to CAD$START to activate the MPP-side routine.



When the Parallel Pascal subroutine has completed its task, control is returned to the VAX via a call to a specially written MCL routine called HALT. Once control has been returned to the VAX, the applications program can interact with the user so that he can select the next function to be performed on the MPP.

When an MPP Shell subroutine is invoked, the VAX side header routine packs any parameters that are needed by the Parallel Pascal routine into a buffer. The header routine then transfers control to the Parallel Pascal routine (via CAD$START), which reads the parameters from the host using the standard MCU Host Memory Read subroutine. These parameters may include indicies that point to a static Parallel Pascal array. Thus, the result of a previous MPP Shell subroutine invocation can be referenced, and results of the present MPP Shell routine can be placed in the static array for use by other routines. A mechanism to reserve static memory via the Parallel Pascal compiler was developed but is not discussed here; interested persons are encouraged to contact the authors.

MPP-side subroutines that are not set up as VAX callable may be called by the Parallel Pascal portion of an MPP Shell subroutine. Thus, the large library of existing Parallel Pascal and MCL subroutines can be used, and eventually integrated into the MPP Shell Library.

A small FORTRAN code example is shown below, which illustrates the simplicity with which a host side applications program can allocate, use and deallocate the MPP.

```
CALL MPP_OPEN

CALL MPP_LOAD(VAX_real, R1)

CALL MPP_LOAD(VAX_imaginary, I1)


CALL FFT(R1,I1, R2, I2)


CALL MPP_UNLOAD(R2, VAX_real)

CALL MPP_UNLOAD(I2, VAX_imaginary)

CALL MPP_CLOSE
```

Where:

VAX_real and VAX_imaginary are 128 x 128 real arrays

R1, I1, R2, and I2 are integers that point to 128 x 128 real arrays in the static Parallel Pascal array

MPP VIRTUAL MEMORY

As mentioned previously, many applications require more than 1/8 k bytes of memory per processor. The MPP Virtual Memory software that was developed allows for the straightforward use of the staging memory as an extension to the array memory. For applications that cannot avoid the use of the stager, the MPP Virtual Memory package handles page swapping between the array memory and the stager. An LRU (Least Recently Used) algorithm was developed to control swapping of data to and from the array.

The static array mentioned above is used as a cache for the MPP Virtual Memory package. The MPP Virtual Memory package allows the programmer to use the stager and the cache memory as a single integerated memory. For the purposes of the present application, a granularity of 32 bits planes per page was chosen. Before an MPP Shell routine begins performing its task, it ensures that the desired page(s) of MPP virtual memory are "paged in" by issuing a call to "MPPVM". MPPVM checks a list of pages that are currently "paged in". If the desired page is not in the cache, MPPVM performs all necessary array-stager data movement to put the page in memory. This may involve "paging out" a page that has not been used recently, in order to make room for the desired page. MPPVM returns the physical page number into which it has placed the desired page. If the desired page is already in memory, MPPVM returns immediately with the appropriate physical page number.

MPPVM is sensitive to the access mode that is required by the caller. Three modes are available: read, write, and update. By specifying the access mode, MPPVM can decide whether a given page needs to be physically moved, and thus prevent unnecessary data movement. The overhead incurred by MPPVM is negligible considering the ammount of time that is required by a single array-stager data move.

NETWORK ACCESS

Transparent network access to the MPP has been established through the use of interprocess mailbox communications supported by DECnet. A process on a remote node invokes a COM file on the MPP VAX which implicitly logs on to the MPP VAX and calls the MPP Shell. Packets are exchanged between the two VAXes to pass data and control information. This technique has been used to support interactive MPP applications accross the network.

CONCLUSION

The MPP is a valuable resource for heavy computational tasks which is needed by the scientific community. The tools that have been presented here promote the rapid implementation of user friendly systems that can access the MPP transparently. The power of the MPP can thus become more readily available to a larger user community as an integrated component of user friendly systems.

N87-26565

MPP PARALLEL FORTH

John E. Dorband
Image Analysis Facility/Code 635
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

MPP Parallel FORTH is a derivative of FORTH-83 and Unified Software Systems' Uni-FORTH. We will describe in this presentation the extension of FORTH into the realm of parallel processing on the MPP. With few exceptions, Parallel FORTH was made to follow the description of Uni-FORTH as closely as possible. Likewise, the parallel FORTH extensions were designed as philosophically similar to serial FORTH as possible. The Massively Parallel Processor (MPP) hardware characteristics, as viewed by the FORTH programmer, will be discussed. Then a description will be presented of how parallel FORTH is implemented on the MPP.

Keywords: FORTH, parallel languages, SIMD, MPP.

INTRODUCTION

The MPP is primarily capable of two types of processing, serial or scalar processing, and parallel processing. The MPP contains an array of 16,384 processing elements(PE's), the array unit. They are all given the same instruction at the same time; thus computing in this array can be viewed as serial processing on a single processor. Yet, processing is actually happening on all 16,384 processors at the same time. Each processor is a bit serial processor with 1024 bits of memory. Thus, the entire array contains 2 million bytes of memory and can be viewed as 1024 bit planes of 128x128 bits each.

The MPP array is a square mesh of processors, where each processor can pass data to its four adjacent processors. The edges of the mesh can be connected in various ways to form several other topologies. These topologies consist of such arrangements as a simple square, a torus, cylinders, or a helix. This communication arrangement allows the programmer to move, simultaneously, as much as 16,384 bits of data 64 processors away from their original source processor. These moves may be in one of four directions - north, west, south, and east.

As well as having a main control unit for scalar processing and an array unit for processing 16,384 elements of data in parallel, the MPP has a staging memory. This memory is the means by which data is moved from the host computer (VAX-11/780) into the array unit memory. The staging memory contains 32M bytes of memory, allowing it to be configured as 16,384 bit planes of 128x128 bits.

Therefore, the MPP, as viewed by the FORTH user, consists of essentially three main components: the main control unit (MCU) (the scalar processor and controller of the array), the array unit (ARU) (for parallel processing of data), and the staging memory (STG) (primarily used for I/O and as a large external bit plane memory).

If every processor had to perform every instruction given to it, it would be of little use as a general purpose computer. Conditional processing alleviates this problem. Conditional processing (such as the execution of an 'IF ... ELSE ... THEN' statement) on the array divides the processor into two groups of processors - those processors for which the condition is true and those for which the condition is false. Since processors can be individually told not to execute the current instruction, the processors for which the condition is true will only execute those instructions between the IF and the ELSE and those processors for which the condition is false will only execute those instructions between the ELSE and the THEN. Thus, through prudent use of conditional statements, the processors can be programmed to perform a range of different functions within the same general time span.

Parallel FORTH is implemented as simply and as straightforward as possible. A Uni-FORTH system is implemented on the MCU. Parallel extensions have been added to the kernel under a new vocabulary called PARALLEL. Context switching has been simplified so that the FORTH word '{' switches to the parallel vocabulary and '}' switches back to the vocabulary that was in use before the switch to the parallel vocabulary. This allows the user to redefine serial words as analogous parallel words under a parallel context, thus making it

PRECEDING PAGE BLANK NOT FILMED

easier for the FORTH user to remember the new parallel words. For example, '+' normally means to add two numbers that are on the data stack, but in the parallel context, '{ + }', means add two 128x128 arrays of numbers on the array stack, which is in the ARU memory.

Two new stacks have been added to parallel FORTH that are not in serial versions of FORTH. These two stacks are the array stack (A) and the mask stack (M). The mask stack is not normally used or seen by the FORTH programmer. It is used to facilitate nested conditional statements, such as 'IF ... ELSE ... THEN' or 'BEGIN ... UNTIL'. The array stack is extensively used by the FORTH programmer, since it is the parallel equivalent of the MCU's data stack. Most operations that can be performed on elements of the data stack have corresponding operations that can be performed in parallel on the array stack, such as +, *, DUP, DROP, and ROT. There are a few other operations that are peculiar to the array stack.

The following sections will discuss in more depth the parallel operations that have been implemented to extend FORTH into the realm of parallelism.

VOCABULARY AND DATA DEFINITION

In MPP Parallel Forth there is a vocabulary called PARALLEL. All new parallel words are in this vocabulary except PE control unit (PECU) primitive words and mask stack operations. As pointed out in the introduction '{' and '}' are used to enter and exit the parallel vocabulary. The following is a definition that will manipulate the MCU data stack:

```
: MULTADD * + ;
```

While the next definition manipulates the ARU array stack:

```
: MULTADD { * + } ;
```

Parallel variables can be allocated in either the array or the staging memory. If a user wants to allocate a 128x128 array of 7-bit values named AR1 in the staging memory, the following is used:

```
7 STG VARIABLE AR1
```

If a user wants to allocate a 128x128 array of 11-bit values in the array memory named AR2, the following is used:

```
11 ARU VARIABLE AR2
```

The definition of parallel constants is similar to defining variables, except the user puts an array on top of the array stack and then executes the statement:

```
13 ARU CONSTANT CON1
```

to create a 128x128 array of 13-bit constants. Likewise vectors and arrays of 128x128 arrays may be defined with VECTOR and MATRIX, respectively. A vector of 20 8-bit 128x128 arrays can be defined with the following statement:

```
20 8 ARU VECTOR VEC1
```

The word ALLOT will allocate variable space in either the stager or the array if it is preceded by the word STG or ARU, respectively. ALLOT is used by all the above-mentioned definition words.

PARALLEL I/O

Parallel files can be stored on the host in either matrix or image format. Each format allows for 8-bit, 16-bit and 32-bit values. A matrix format file contains multiple arrays of 128x128 values. An image format file contains multiple images of 512x512 values. The following command:

```
CHANA IMAGE8 OPEN WHAT.DAT
```

opens the file 'what.dat' as an image file of images with 8-bit values. LOAD is then used to read the matrix or image into a previously defined staging memory array. An image from an image file of 8-bit values should only be loaded into a VECTOR or MATRIX that has at least 8x16 or 128 bits allocated to it. The command to read a matrix into a stager array is:

```
V1 3 LOAD
```

This loads the third matrix of the current file into variable V1. To store an image into a file, the word STORE is used (i.e., V1 3 STORE).

MEMORY OPERATIONS

Memory operations are used to move data between the three MPP memories: the MCU, the ARU, and the staging memory. The word '@' fetches arrays from array variables in the stager and the ARU memory and puts them on the array stack. The word '!' stores an array from the array stack into an array variable in the stager or ARU memory. The word 'SCALAR' takes a value from the data stack in the MCU memory, broadcasts it to all PE's, and produces an array on top of the array stack that has the same value for all elements of the array. Also, when the context is the parallel vocabulary, any literals are compiled into constants that will be sent to the top of the array stack as a scalar value during execution. Operations such as GMAX, GMIN, and GOR can reduce an array of values into a scalar value that can be put onto the data stack.

ARRAY STACK OPERATIONS

Most array manipulation occurs on the array stack. The array stack consists of a stack of descriptors in MCU memory and the actual bit plane stack in the ARU memory. The array stack is manipulated by operations very similar to those used on the data stack. These operations consist of words such as DUP, DROP, SWAP, OVER, ROT, PICK, and ROLL. In addition to the standard stack operations there are also operations that are peculiar to the array stack. They consist of the following words: -NDROP, NDROP, A@, >A, ZERO, EXTRACT, SLIDE, EXG, CROSS, and TOPOLOGY.

NDROP drops the top n elements of the array stack. -NDROP skips the first n1 elements of the stack and drops the next n2 elements of the array stack. 'A@' copies the descriptor off of the MCU array stack onto the data stack. A parallel array descriptor consists of two values: the address of the least significant bit plane of the array(LSB) and the number of bit planes in the array(LEN). '>A' creates an array of n bit planes on the array stack, where n is taken from the top of the data stack. ZERO is the same as '>A' except the bit planes are initialized to zero. EXTRACT extracts a field of bits from the second element of the array stack and inserts it into a field of the same size in the top element of the stack. SLIDE slides the top element of the array stack across the array of PE's. EXG exchanges data in the top elements of the array stack among PE's of the ARU. CROSS exchanges data from the top elements of the array stack with the second elements of the array in different PE's of the ARU. The TOPOLOGY operation changes the topology of the ARU.

ARITHMETIC, LOGIC, AND COMPARISON OPERATIONS

All the operations in this section deal primarily with the elements on the top of the array stack. Basically they are analogous to the corresponding operations that operate on the top of the data stack. The difference is that operations on the array stack perform 16,384 operations at the same time instead of one at a time and values on the array stack can have variable numbers of bits instead of a fixed number such as 8, 16, or 32.

Normally operations on the data stack are either single or double precision. On the array stack, however, operations are classified as either fixed or variable precision. A fixed precision operation requires that both operands have the same length and that their result is the same length. A variable precision operation may operate on operands whose lengths are different. The result of such operations has a length that is dependent on both the specific operation and the length of the operands. All basic operations discussed here have a fixed precision operation. Only a few operations have both a fixed and a variable precision form of operation. These operations are +, -, *, /, MOD, and /MOD. Their variable precision forms are ~+, ~-, ~*, ~/, ~MOD, and ~/MOD.

The result of a ~+ or a ~- operation has a length equal to one plus the maximum of the two operands. The result of a ~* operation has a length equal to the sum of the length of the two operands. The length of result of a ~/ operation is the difference between the length of the dividend operand and the length of the divisor operand. Note that the length of the dividend must be larger than that of the divisor. The result of the ~MOD operation has a length equal to the length of the divisor operand. Since the result of the ~/MOD operation is the result of the ~/ operation followed by the ~MOD operation, the lengths of the results are the same as described for ~/ and ~MOD.

The fixed precision only operations are MAX, MIN, ABS, NEGATE, 1+, 1-, 2/, 2*, AND, OR, XOR, and NOT. Three special operations find the aggregate result and place it on the data stack. These global operations are global maximum(GMAX), global minimum(GMIN), and global or(GOR).

Comparison operations differ slightly from the other operations in this section in that they result in a value of length 1. These operations are <, =, >, 0<, 0=, and 0>.

CONTROL OPERATIONS

Control operations cause certain portions of code to be executed on some data and not on others. Parallel control is quite different from serial control. In serial control, condition evaluation determines whether or not a certain piece of code will be executed. In parallel control, both the code corresponding to the true condition and the false condition may have to be executed. Some of the processors must be turned off during the execution of the code for the true condition, then turned on for the execution of the code for the false condition. This is accomplished with a mask bit. It is set to one in processors whose data satisfy the condition, and to zero in those whose data does not. Thus only those processors that satisfy the condition execute the code for the true condition. The mask bit is then complemented and only those processors that did not satisfy the condition will execute the code for the false condition. As with execution of serial conditions, parallel conditions can be nested. Therefore, there is a mask stack. Mask stack primitive operations are used to implement the operations in this section.

The basic conditional structure is the IF ... ELSE ... THEN statement. The IF word duplicates the top element on the mask stack, takes the least significant bit of the top element of the array stack, and ands it to the top element on the mask stack. The ELSE word complements the top element of the mask stack. And the THEN word drops the top element of the mask stack.

The parallel conditional loop structure is also somewhat unusual. It continues to execute as long as there is a processor that has not met the condition to terminate the loop. The two types of loops are the BEGIN... UNTIL and the BEGIN ... WHILE ... REPEAT. The BEGIN word duplicates the top element of the mask stack. The REPEAT word marks the end of the loop. The WHILE word ands the least significant bit of the top element of the array stack to the top element of the mask stack and terminates the loop if no processor has the top element of the mask stack equal to one. The UNTIL word is the same as WHILE except the least significant bit of the top element of the array stack is complemented before it is anded to the top element of the mask stack.

Note that only certain operations are maskable. Therefore, one should be aware that operations may execute when the processor was masked out because the operation was not maskable. Generally, only operations that do not change the number of elements on the array stack or the order of the elements on the array stack are maskable. Thus, most stack manipulation operations and two operand operations are not maskable. See the MPP Parallel FORTH Word Reference for more specific details.

PECU AND MASK STACK PRIMITIVES

The PECU and mask stack primitives are not meant to be used by the general FORTH programmer. They are used by the primary parallel FORTH words to initiate actions to be performed in the ARU. If it is necessary to use them, they are described in the MPP Parallel FORTH word reference.

MPP PARALLEL FORTH WORD REFERENCE

Context Changing Words

PARALLEL

'PARALLEL' is the vocabulary that contains all the words that act on data in the MPP array unit.

{

This changes the context of word searches to the 'PARALLEL' vocabulary.

}

This returns the context to that specified prior to the change to the 'PARALLEL' vocabulary.

Arithmetic Words

+ (A: a1(n) a2(n) -- A: a3(n))

This adds a1 and a2, which are the same size and produces a3, which is that size.

- (A: a1(n) a2(n) -- A: a3(n))

This subtracts a2 from a1, which are the same size and produces a3, which is that size.

* (A: a1(n) a2(n) -- A: a3(n))

This multiplies a1 by a2, which are the same size and produces a3, which is that size.

/ (A: a1(n) a2(n) -- A: a3(n))

This divides a1 by a2, which are the same size and produces a3, which is that size.

MOD (A: a1(n) a2(n) -- A: a3(n))

This divides a1 by a2, which are the same size and produces the remainder a3, which is that size.

/MOD (A: a1(n) a2(n) -- A: a3(n) a4(n))

This divides a1 by a2, which are the same size and produces a3, the remainder, and a4, the quotient, which are of that size.

MAX (A: a1(n) a2(n) -- A: a3(n)) {maskable}

This finds the maximum of a1 and a2, which are the same size and produces a3, which is that size.

GMAX (A: a1(n) -- S: m) {maskable}

This finds the global maximum of the a1 and places it on the data stack as a scalar value.

MIN (A: a1(n) a2(n) -- A: a3(n)) {maskable}

This finds the minimum of a1 and a2, which are the same size and produces a3, which is that size.

GMIN (A: a1(n) -- S: m) {maskable}

This finds the global minimum of the a1 and places it on the data stack as a scalar value.

ABS (A: a(n) -- A: a(n)) {maskable}

This finds the absolute value of 'a' and replaces 'a' on the stack.

NEGATE (A: a(n) -- A: a(n)) {maskable}

This finds the 2's complement of the value of 'a' and replaces 'a' on the stack.

1+ (A: a(n) -- A: a(n)) {maskable}

This increments the value of 'a' and places it back on the stack.

1- (A: a(n) -- A: a(n)) {maskable}

This decrements the value of 'a' and places it back on the stack.

2/ (A: a(n) -- A: a(n)) {maskable}

This shifts a(n) to the right.

2* (A: a(n) -- A: a(n)) {maskable}

This shifts a(n) to the left.

~+ (A: a1(n) a2(m) -- A: a3(max(n,m)+1))

This adds a1 to a2 and produces a result, a3, which has a size that is the maximum of the sizes of a1 and a2, plus 1.

~- (A: a1(n) a2(m) -- A: a3(max(n,m)+1))

This subtracts a1 from a2 and produces a result, a3, which has a size that is the maximum of the sizes of a1 and a2, plus 1.

~* (A: a1(n) a2(m) -- A: a3(n+m))

This multiplies a1 by a2 and produces a result, a3, which has a size that is the sum of the sizes of a1 and a2.

~/ (A: a1(n) a2(m) -- A: a3(n-m))

This divides a1 by a2 and produces a result, a3, which has a size that is the difference of the sizes of a1 and a2.

~MOD (A: a1(n) a2(m) -- A: a3(m))

This divides a1 by a2 and produces the remainder, a3, which has a size of a2.

~/MOD (A: a1(n) a2(m) -- A: a3(n-m) a4(m))

This divides a1 by a2 and produces a remainder, a3, which has a size the same as a2 and a quotient, a4,which has a size that is the difference of the sizes of a1 and a2.

Logical Words

AND (A: a1(n) a2(n) -- A: a3(n))

This ands a1 and a2, which are the same size and produces a3, which is that size.

OR (A: a1(n) a2(n) -- A: a3(n))

This ors a1 and a2, which are the same size and produces a3, which is that size.

GOR (A: a1(n) -- S: m) {maskable}

This finds the global 'or' of the a1 and places it on the data stack as a scalar value.

XOR (A: a1(n) a2(n) -- A: a3(n))

This xors a1 and a2, which are the same size and produces a3, which is that size.

NOT (A: a(n) -- A: a(n)) {maskable}

This finds the complement value of 'a' and places it back on the stack.

Comparison Words

< (A: a1(n) a2(n) -- A: a3(1))

This determines if a1 is less than a2, and produces a bit plane that is 1 where it is true and 0 where it is false.

= (A: a1(n) a2(n) -- A: a3(1))

This determines if a1 is equal to a2, and produces a bit plane that is 1 where it is true and 0 where it is false.

> (A: a1(n) a2(n) -- A: a3(1))

This determines if a1 is greater than a2, and produces a bit plane that is 1 where it is true and 0 where it is false.

0< (A: a1(n) -- A: a2(1))

This determines if a1 is less than 0, and produces a bit plane that is 1 where it is true and 0 where it is false.

0= (A: a1(n) -- A: a2(1))

This determines if a1 is equal to 0, and produces a bit plane that is 1 where it is true and 0 where it is false.

0> (A: a1(n) -- A: a2(1))

This determines if a1 is greater than 0, and produces a bit plane that is 1 where it is true and 0 where it is false.

Stack Operation Words

DUP (A: a(n) -- a(n) a(n))

Duplicates the top element on the array stack.

DROP (A: a(n) --)

Drops the top element on the array stack.

NDROP (S: n A: a(m)...a(p) --- A: a(m) a(q))

Drops the top n elements of the array stack.

-NDROP (S: n1 n2 A: a1(p1)...a2(p2)...a3(p3)
--- A: a1(p1) a2(p2)...a3(p3))

Skips the first n1 elements of the array stack and drops the next n2 elements of the array stack.

SWAP (A: a1(n) a2(n) -- A: a2(n) a1(n))

Swaps the top two elements on the array stack.

OVER (A: a1(n) a2(n) -- A: a1(n) a2(n) a1(n))

Copies the second element on the array stack to the top of the stack.

ROT (A: a1(n) a2(n) a3(n) -- A: a2(n) a3(n) a1(n))

Moves a1 to the top of the array stack.

PICK (S: m A: a1(n) ... a2(n)
-- A: a1(n) ... a2(n) a1(n))

Copies the mth element of the stack to the top of the stack. (1 PICK is the same as OVER.)

ROLL (S: m A: a1(n) ... a2(n) a3(n)
-- A: ... a2(n) a3(n) a1(n))

Moves the mth element of the stack to the top of the stack. (3 ROLL is the same as ROT.)

DEPTH (-- S: n)

Returns the number of elements on the array stack.

A@ (A: a(n1) --- S: n2 n1)

Copies the first descriptor on the array stack onto the data stack.

>A (S: n --- A: a(n))

Creates an element on top of the array stack that has n bit planes.

ZERO (S: n --- A: a(n))

Create an element of size n that has a value of zero onto the top of the array stack.

EXTRACT (S: m1 m2 n A: a1(n1) a2(n2)
--- A: a1(n1) a2(n2))

Extracts a field of n bits of a1(n1) starting at m1 and places it in a2(n2) starting at m2.

SLIDE (S: n d A: a(p) -- A: a(p))

Slides 'a' n PE's in the direction designated by d. East if d=0, west if d=1, south if d=2, and north if d=3.

EXG (S: m1 m2 n d A: a(n) -- A: a(n))

Exchanges elements of 'a' n PE's apart in the direction designated by d. East/west if d=0, south/north if d=2. The addresses of mask bit planes are m1 and m2. The mask m1 determines which PE's accept data during the east or south portion of the move and m2 determines which PE's accept data after the west or north portion of the move.

CROSS (S: m1 m2 n d A: a1(n) a2(n)
-- A: a1(n) a2(n))

Exchanges elements of a1 with a2 n PE's apart in the direction designated by d. East/west if d=0, south/north if d=2. Elements of a1 move to the east or south and elements of a2 move to the west or north. The addresses of the mask bit planes are m1 and m2. The mask m1 determines which PE's accept data during the east or south portion of the move and m2 determines which PE's accept data after the west or north portion of the move.

TOPOLOGY (S: n ---)

The number n designates the topology of the array when an EXG, SLIDE, or CROSS is performed.

Topology	North/South Connection	East/West Connection
0	None	None
1	None	Cylinder
2	Cylinder	None
3	Cylinder	Cylinder
4	Open-spiral	None
5	Open-spiral	Cylinder
6	Closed-spiral	None
7	Closed-spiral	Cylinder

Memory Operation Words

@ (S: m --- A: a(n)) {maskable}

Moves an array variable described by a descriptor at address m onto the array stack.

! (S: n A: a(n) ---) {maskable}

Moves an array from the array stack into an array variable described by a descriptor at address m.

SCALAR (S: <scalar value> --- A: a(n))
{maskable}

Broadcasts a scalar value into array 'a' of all PE's.

LITERAL

Compiles a constant into a word that will be placed onto the array stack during execution, or will immediately place it on the stack during interpretation.

LIT

This is the execution time routine used to place a constant, compiled into the code, onto the array stack.

DESC (S: n --- S: n2 n1)

Fetches the descriptor at address n and places it on the data stack. The address of the least significant bit plane (LSB) of the variable is n2 and n1 is the size of the variable.

Control Words

IF (A: a(n) M: m1 --- M: m1 M: m2)

Creates a new layer on the mask stack that is the result of anding the least significant bit plane of 'a' and m1.

ELSE (M: m --- M: m)

Complements the value of the top element of the mask stack.

THEN (M: m ---)

Drops the top element of the mask stack.

BEGIN (M: m1 --- M: m1 m1)

Duplicates the top element of the mask stack.

WHILE (A: a(n) M: m1 --- M: m1)
or (A: a(n) M: m1 ---)

Ands the least significant element of 'a' and m1. If no element of m1 is equal to 1, the loop is terminated.

REPEAT

Marks the end of a BEGIN...WHILE...REPEAT loop.

UNTIL (A: a(n) M: m1 --- M: m1 M: m2)
or (A: a(n) M: m1 ---)

Ands the complement of the least significant element of 'a' and m1. If no element of m1 is equal to 1, the loop is terminated.

I/O Words

MATRIX8
MATRIX16
MATRIX32

File types for files that contain 128x128 arrays of 8, 16, or 32-bit values.

IMAGE8
IMAGE16
IMAGE32

File types for files that contain 512x512 images of 8, 16, or 32-bit values.

LOAD (S: n2 n1 ---)

Loads an array n1 or image n1 from the currently opened file into the designated bit plane described by the descriptor at address n2.

STORE (S: n2 n1 ---)

Stores the designated bit planes described by the descriptor at address n2 into an array n1 or image n1 of the currently opened file.

Defining Words

VARIABLE (S: n f ---)

Allocates an n bit plane variable array in either the stager or the array.

CONSTANT (S: n f A: a(n) ---)

Allocates an n bit plane constant array in either the stager or the array and loads it with the top element of the array stack.

VECTOR (S: m n f ---)

Allocates a vector of m n bit values in either the stager or the array.

MATRIX (S: m1 m2 n f ---)

Allocates an m1xm2 matrix of n bit values in either the stager or the array.

Compiler Words

ALLOT (n f ---)

Allocates n bit planes in either the array (ARU) or the stager (STG).

ARU

Indicates that the desired variable will be allocated in the array.

STG

Indicates that the desired variable will be allocated in the stager.

PECU Primitive Words

PECU (S: <PECU address> --)

The word 'PECU' takes an address off the MCU data stack and places it in register 'SPE', which starts the PECU at that address.

S>C (S: <64 bit scalar> <LSB of scalar> ---)

The word 'S>C' loads the LSB of a scalar into PE0. The 64-bit scalar value will be loaded into the common register from the data stack.

C>S (--- S: <64 bit scalar> <LSB of scalar>)

The word 'C>S' stores the 64-bit return register A value on the data stack followed by the value 64.

A>PE2 (A: <descriptor> ---)
A>PE4 (A: <descriptor> ---)
A>PE6 (A: <descriptor> ---)

Takes 2 descriptors from the array stack and places them into registers PE2-PE3, PE4-PE5, or PE6-PE7, respectively. Each descriptor consists of a 16-bit LSB and a 16-bit size.

S>PE2 (S: n1 n2 ---)
S>PE4 (S: n1 n2 ---)
S>PE6 (S: n1 n2 ---)

Takes 2 words from the data stack and places them into registers PE2-PE3, PE4-PE5, or PE6-PE7, respectively.

Mask Stack Operations

A>M (A: a(n) M: m1 --- M: m1)

Ands the least significant bit plane of a(n) to m1. The mask stack pointer is maintained in PE1.

M>A (M: m1 --- M: m1 A: a(1))

Copies the top bit plane of the mask stack onto the top of the array stack.

MDROP (M: m1 ---)

Drops a mask from the mask stack.

MDUP (M: m1 --- M: m1 m1)

Duplicates top element on mask stack.

ACKNOWLEDGEMENTS

I would like to express my appreciation to Dick Fahey, Arne Henden, Dan Klinglesmith, and Archie Warnock for their advice and consultation on the FORTH philosophy and to Lisa Hamet for her help in the coding and implementation of the Uni-FORTH kernel on the MPP.

REFERENCES

1. Brodie, Leo, "Starting FORTH," Prentice-Hall, Inc., Englewood Cliffs, NJ., 1981.

2. Brodie, Leo, "Thinking FORTH," Prentice-Hall, Inc., Englewood Cliffs, NJ., 1984.

3. Henden, Arne, "Uni-FORTH User's Guide," Unified Software Systems, Columbus OH, 1985.

APPENDIXES

PRECEDING PAGE BLANK NOT FILMED

APPENDIX A

Index of Authors

PRECEDING PAGE BLANK NOT FILMED

APPENDIX A

Index by Authors

Abeles, Jim 161
Barnden, Dr. John A. 59
Batcher, Dr. Ken 151
Bielefield, Dr. Michael J. 111
Camillo, P. J. 19
Carmichael, Dr. Gregory R. 165
Cho, Seog Y. 165
Cohen, David M. 165
Devaney, J. E. 19
Davis, Dr. Edward W. 135
Dorband, Dr. John E. 211, 275
Earl, Dr. James A. 197
Fatoohi, R. 3
Fiebrich, Dr. Rolf D. 239
Gledhill, Dr. I. M. A. 37
Gough, Michael L. 217, 271
Greenstadt, Eugene W. 223
Grosch, Dr. Chester E. 3
Gurney, Dr. Robert J. 19
Harten, Paul 47
Hastings, Dr. Harold M. 69
Heap, Dr. Sara R. 99
Ida, Dr. Nathan 257
Irving, A. R. 19
Janetzke, R.W. 185
Jordan, Karen F. 223
Koga, J. 185
Liebrock, Lorie M. 193
Lin, Dr. Chin S. 185
Lindler, Donald J. 99
McAllister, Dr. David F. 135
McAnulty, Dr. Michael A. 227
McKurdy, Kathy J. 275
Mountain, Dr. Raymond D. 53
O'Donnell, Dr. John T. 75
Oguztuzun, Mehmet H. 165
Opsahl, Dr. Torstein 249
Ozga, Martin 13
Parkinson, Dr. Dennis 205
Pol, Sanjay 135
Ramapriyan, Dr. H. K. 121
Reif, Dr. John H. 249
Seiler, Edward J. 121
Storey, Dr. L. R. Owen 37
Strong, Dr. James P. 141
Suarez, Dr. Max J. 161
Sullivan, Dr. Francis 53
Suranyi, Dr. Peter 47

PRECEDING PAGE BLANK NOT FILMED

Thring, A. L. 185
Tilton, Dr. James C. 171
Treinish, Lloyd A. 217
Udawatta, Kapila 257
Von Laven, Dr. Scott A. 193
Wainer, Michael S. 227
Waner, Dr. Stefan 69
White, Dr. Richard L. 29
Wildenhain, David W. 217,275
Wolfram, Dr. Stephen 87
Wunderlich, Dr. Marvin C. 265
Yin, Dr. Lo I. 111

APPENDIX B
Technical Summary

TECHNICAL SUMMARY

THE MASSIVELY PARALLEL PROCESSOR

The Massively Parallel Processor (MPP), an advanced computer architecture termed single-instruction stream multiple-data stream (SIMD), shows promise of delivering enormous computational power and at lower cost than other existing architectures. Its computational element, the array unit, consists of a 128 by 128 array of small 1-bit processors, each containing 1,024 bits of local memory, and having nearest neighbor connectivity. A secondary storage unit, the staging memory, holds 32 megabytes of data and connects to the array memory via an 80-megabyte-per-second data path. An array control unit broadcasts control signals to all processors in the array unit. The MPP is a back-end processor for a VAX-11/780 host, which supports its program development and data needs. (See Figure 1.)

The MPP was built for the Goddard Space Flight Center by Goodyear Aerospace Corporation and delivered in May 1983. At that time, the construction of a digital processor using the very high degree of parallelism embodied in the MPP had not been previously attempted.

MPP Software

Since its delivery to Goddard, an extensive language system and a unique operating system have been implemented for the MPP. Dozens of teams of scientific investigators who are developing, testing, and running parallel algorithms rely on this system software repetoire daily. The initial high-level language implemented in 1983 was Parallel Pascal. This language was designed to be independent of computer architecture, thus allowing portability of applications programs between diverse parallel computers having Parallel Pascal compilers. Experience gained in the development and use of this approach revealed that the MPP's 128- by-128 square grid architecture could not easily be hidden from the programmer by using current compiler writing technology. A modified language, MPP Pascal, was then implemented that is architecturally dependent, possessing important semantic features that allow the programmer to make very efficient use of the hardware's capabilities. The MPP Pascal compiler is capable of producing highly optimized code and is flexible enough to allow easy modification. The MPP is also programmable in assembly language.

The MPP operating system provides interactive debugging aids in addition to support for running applications code. The software that performs these tasks is shared by all MPP users, greatly reducing the demand on the host's main memory. The debugging aids include performance monitoring, error reporting of MPP hardware detected faults, breakpointing, single-step and status display. A first-come-first-served queue is the central arbiter controlling user access to the MPP.

All MPP applications programs must be prepared as two parts. One part runs in the MPP control unit; the other part runs on the host. They are linked together through a message passing system. A master/slave control relationship exists between the MPP and the host. The host resident program is the highest level of control. This program interacts with the user and starts MPP programs. MPP programs, in turn, use the host as an I/O server, directly accessing the host's disk and image analysis terminals through an extensive set of I/O service routines.

A device driver that communicates directly with the MPP hardware runs at the lowest level in the host operating system. This driver is the hub of the entire system, controlling the execution of programs in the MPP, as well as the flow of data throughout the system. The bulk of the operating system interacts directly with

PRECEDING PAGE BLANK NOT FILMED

this device driver to accomplish tasks in support of a running application such as initializing the hardware, loading programs, starting and stopping programs, reading and writing data, and delivering messages between running programs in the host and the MPP.

A number of libraries of computational subroutines are supported. One type holds more than 270 microcoded subroutines that define the actual instruction set of the MPP. These include the basic arithmetic and transcendantal functions, as well as multi-precise arithmetic and special user-written instructions. A second type of library holds MPP Pascal callable subroutines including fast fourier transforms, a random number generator, a sort computation package, linear algebra routines, and utility programs. Another set of libraries holds I/O subroutines that control the movement of data within the system.

For many applications, having only 1,024 bits of memory available to each of the 16,384 MPP processors has been a serious constraint. The memory chip technology available in 1980 when the system was designed imposed this limitation. As an alternative to an expensive hardware upgrade, a Bit-Plane I/O software package was developed that treats the staging memory as individual bit-planes. A system was implemented that provides each processor with 16K bits of virtual array memory. The penalty is an increase in memory access time from .1 microseconds to 25 microseconds per bit plane. However, many applications benefit from this virtual memory, as they effectively overlap computation with data transfer. In addition to Bit-Plane I/O, another system, SMM I/O, gives the user access to the powerful data reformatting capabilities of the staging memory.

Two MPP simulation environments have been developed and distributed to remoted user sites. One is the MPP Simulator, which supports the development, testing, and refinement of MPP Pascal, or assembly language applications programs on any VAX operating under VMS. It allows a user the convenience of a local dedicated MPP that doesn't need be shared with other users. In addition, its use at remote sites off-loads program preparation work from the MPP/VAX system at Goddard. Code that runs on the Simulator will run on the MPP after adjusting any references to the size of the array unit (usually simulated as a 16 x 16 array to speed execution).

A second simulation tool, the Parallel Pascal Translator, takes Parallel Pascal source code as input and produces equivalent serial Pascal source code as output. The serial Pascal can be compiled and executed using a standard Pascal compiler system. The Translator allows the development, testing, and refinement of applications programs on most computers that have a Pascal compiler.

MPP HARDWARE

Array Unit

The Array Unit, (the 128 x 128 processing element (PE) array) supplies the MPP's computational power. Each PE has a local 1,024-bit random access memory and is connected to its four nearest neighbors--north, south, east, and west. Opposite array edges can be connected together to form either a plane, a horizontal cylinder, a vertical cylinder, or a torus. Arithmetic and logic in each PE are performedin bit-serial manner. All operands are located in the 1,024-bit local memory. The cycle time is 100 nanoseconds. Table 1 shows the raw computing speeds for selected arithmetic operations. The data-bus states of all 16,384 PEs are combined in a tree of inclusive-or logic elements whose single wire output is used in the Array Control Unit for operations such as finding the maximum or minimum value in parallel of an array.

MPP Processing Element: A single PE is shown functionally in Figure 2. The P-register, together with its input logic, performs all Boolean logic functions on two variables and can also receive data from the P-register in any one of its four nearest neighbors. The A-, B-, and C-registers, the shift register, and associated logic form an arithmetic unit. The G-register controls masking of arithmetic, logic, and routing operations. (Unmasked operations are performed in all PEs. Masked operations are only performed in those PEs where the G-register is set.) The S-register is used to shift data to and from the Staging Memory without disturbing PE operations. A custom integrated circuit (IC) holds eight PEs, exclusive of the 1,024 bits of random access memory, which is on a separate IC chip. A one bit wide bidirectional databus connects the memory and the internal components of the PE.

Microcircuit Technology: All components of eight PEs, exclusive of random access memory, are packaged on a custom integrated circuit. This chip, containing a 2-row by 4-column array of PEs, uses high speed complementary MOSFET (HCMOS) technology.

Array Control Unit

The Array Control Unit broadcasts control signals and memory addresses to all PEs in the Array Unit and receives Array Unit status bits. It is designed to perform bookkeeping operations (address calculation, loop control, branching, subroutine-calling, etc.), and control the Array Unit simultaneously. It contains three parts: (1) the Main Control Unit, (2) the PE Control Unit, and (3) the I/O Control Unit.

The Main Control Unit executes the application program stored in its program memory. It performs the scalar arithmetic operations required, calls the PE Control Unit for all array logic and array arithmetic operations, and calls the I/O Control Unit for all I/O operations. Both sets of calls are queued to await execution while the Main Control Unit moves on to generate other calls.

The PE Control Unit generates all Array Unit instructions except those pertaining to the S-register (data I/O). It executes microcoded routines stored in its program memory to perform all array operations required by applications programs.

The I/O Control Unit controls the shifting of I/O data through the Array Unit S-registers, as well as the transfer of I/O data between the S-registers and the Array Unit memory. It executes I/O channel control programs stored in the Main Control Unit's program memory.

Staging Memory

The MPP system includes a Staging Memory for buffering Array Unit data. This memory provides both the "corner turning" function, which converts conventional byte- or word-oriented data into the bit-plane form needed by the Array Unit, and the "multi-dimensional access" function, which allows large multi-dimensional arrays of data located in the Staging Memory to be read out or written in along arbitrary orderings of array dimensions. The current capacity of the Staging Memory is 32 megabytes and is upgradable to 64 megabytes.

Data moves between the Array Unit and the Staging Memory via 128 parallel lines. The upper limit on the transfer rate is 1.28 billion bits/second. Goddard's MPP currently supports .64 billion bits/second. Data movement in both directions can be overlapped with processing.

Host Processor

A DEC VAX-11/780 computer manages data flow between MPP units, loads programs into the Control Unit, executes system test and diagnostic routines, and provides program development facilities. The MPP is interfaced to the VAX through a 5 megabytes/second DR-780 channel. Remote access to the VAX is provided through ARPANET, SPAN, TELENET, BITNET, and dial-in.

Additional Information

MPP users are selected from those who submit proposals in response to the open-ended NASA Space Science and Applications Notice: "Computational Investigations Utilizing the Massively Parallel Processor (MPP)." Thirty-nine scientific investigators from across the United States currently belong to the MPP Working Group. For additional information, interested parties should contact James Fischer, Mail Code 635, Goddard Space Flight Center, Greenbelt, Maryland, 20771, or by phone at (301) 286-3464.

References

Batcher, K.E., "Design of a Massively Parallel Processor," *IEEE Transactions on Computers*, Vol. C29, September 1980, 836-840.

The Massively Parallel Processor, MIT Press, August 1985, J. L. Potter, ed.

Reeves, A.P., "Parallel Pascal: An Extended Pascal for Parallel Computers," *Journal of Parallel and Distributed Computing*, Vol. 1, 1984, 64-80.

Strong, J.P., "Basic Image Processing Algorithms on the Massively Parallel Processor," *Multicomputers and Image Processing*, New York, 1982, 47-85.

FUNCTIONAL UNITS OF ONE PE

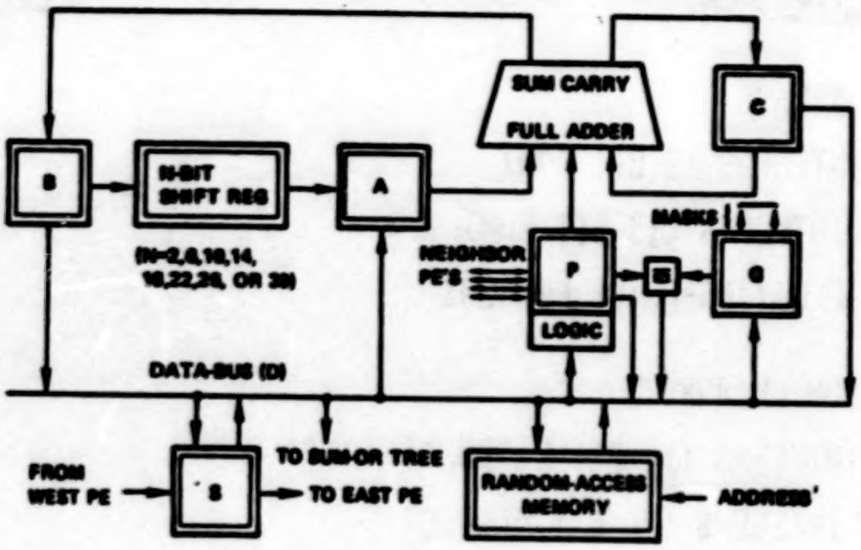

SUM CARRY
FULL ADDER
C
B
N-BIT SHIFT REG
A
(N=2,6,10,14,
18,22,26, OR 30)
MASKS
NEIGHBOR PE'S
P
E
G
LOGIC
DATA-BUS (D)
FROM WEST PE
S
TO SUM-OR TREE
TO EAST PE
RANDOM-ACCESS MEMORY
ADDRESS'

Figure 1

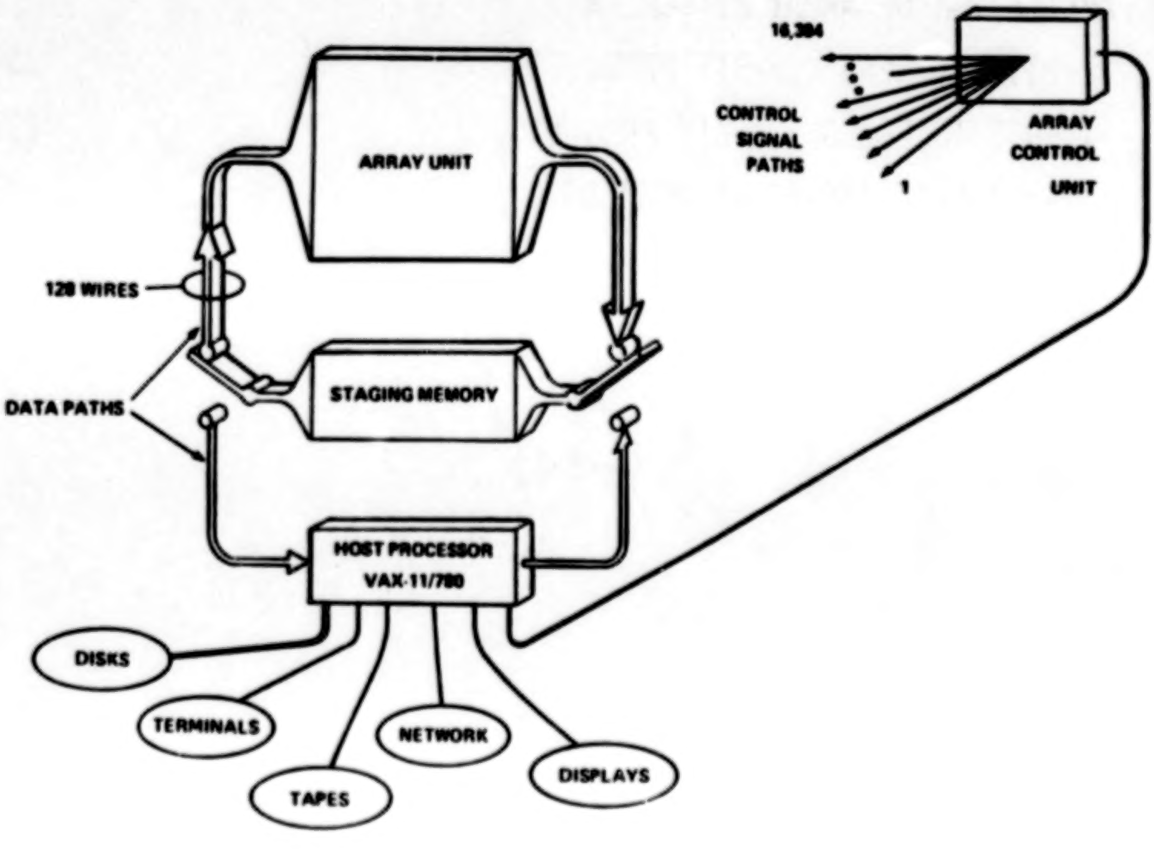

16,384
CONTROL SIGNAL PATHS
1
ARRAY CONTROL UNIT
ARRAY UNIT
128 WIRES
DATA PATHS
STAGING MEMORY
HOST PROCESSOR
VAX-11/780
DISKS
TERMINALS
NETWORK
DISPLAYS
TAPES

Figure 2

SPEED OF TYPICAL OPERATIONS

OPERATION	EXECUTION SPEED, MOPS*
ADDITION OF ARRAYS	
8-BIT INTEGERS (9-BIT SUM)	6553
12-BIT INTEGERS (13-BIT SUM)	4428
32-BIT FLOATING-POINT NUMBERS	430
MULTIPLICATION OF ARRAYS	
8-BIT INTEGERS (16-BIT PRODUCT)	1861
12-BIT INTEGERS (24-BIT PRODUCT)	910
32-BIT FLOATING POINT NUMBERS	216
MULTIPLICATION OF ARRAY BY SCALAR	
8-BIT INTEGERS (16-BIT PRODUCT)	2340
12-BIT INTEGERS (24-BIT PRODUCT)	1260
32-BIT FLOATING-POINT NUMBERS	373

*MILLION OPERATIONS PER SECOND

Table 1

APPENDIX C
Symposium Attendees

LIST OF ATTENDEES
FIRST SYMPOSIUM ON THE FRONTIERS OF MASSIVELY PARALLEL SCIENTIFIC COMPUTATION

MR. JIM ABELES
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-5205

MR. AJIT K. AGRAWAL
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

DR. NICK ALEXANDRIDIS
GEORGE WASHINGTON UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE,
WASHINGTON, DC 20052

MR. BRUCE H. ALPER
CORPORATE FINANCE ASSOCIATES
6600 PEACHTREE DUNWOODY ROAD
BUILDING 300, SUITE 670
ATLANTA, GA 30328
(404) 399-5633

DR. MICHAEL ANDREWS
SPACE TECH CORPORATION
215 EAST OAK STREET
FORT COLLINS, CO 80524
(303) 484-9903

MR. JAMES BAILEY
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MARION BARANIECKI
MARTIN MARIETTA LABS
1450 SOUTH ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

DR. JOHN A. BARNDEN
INDIANA UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
BLOOMINGTON, IN 47405
(812) 335-2136

DR. KENNETH E. BATCHER
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-4511

MR. DAVID BAZELL
APPLIED RESEARCH CORPORATION
8201 CORPORATE DRIVE
LANDOVER, MD 20785
(301) 286-3942

DR. ISABEL M. BEICHL
745 EAST 36 STREET
BALTIMORE, MD 21218
(301) 921-3855

LES BERKE
NASA-LEWIS RESEARCH CENTER
MAIL STOP 49-7
CLEVELAND, OH 44135
(216) 433-3212

DR. DIETER BILITZA
SCIENCE APPLICATIONS RESEARCH
NASA-GODDARD SPACE FLIGHT CENTER
CODE 633
GREENBELT, MD 20771
(301) 286-9536

DR. GABRIEL P. BISCHOFF
DIGITAL EQUIPMENT CORPORATION
77 REED ROAD
HL02-2/H13
HUDSON, MA 01749
(617) 568-6151

MR. DICK BISHOP
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVAP.D
LANHAM, MD 20706
(301) 794-5150

MR. CHARLES BOHN
NASA-GODDARD SPACE FLIGHT CENTER
CODE 635
GREENBELT, MD 20771
(301) 286-7980

MS. BARBARA E. BOOTHMAN
SASC TECHNOLOGIES
NASA-GODDARD SPACE FLIGHT CENTER
GREENBELT, MD 20771
(301) 286-3625

MS. KEIRA BROMBERG
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MR. LEE E. BROTZMAN
STI INCORPORATED
NASA-GODDARD SPACE FLIGHT CENTER
CODE 681
GREENBELT, MD 20771
(301) 286-3625

MR. TIM BUSSE
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5194

MR. PETER J. CAMILLO
NASA-GODDARD SPACE FLIGHT CENTER
CODE 624
GREENBELT, MD 20771
(301) 286-7099

MR. SALVATORE A. CANGIALOSI
U.S. PATENT AND TRADEMARK OFFICE
742 SOUTH 26TH STREET
ARLINGTON, VA 22202
(703) 557-3963

MR. ROBERT M. CAREY
NASA HEADQUARTERS
NOAA SATELLITE RESEARCH LAB
2205 MOUNT TABOR ROAD
GAMBRILLS, MD 21054
(301) 763-4240

PRECEDING PAGE BLANK NOT FILMED

B. F. CAVINESS
UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER AND
 INFORMATION SCIENCES
NEWARK, DE 19716
(302) 451-2711

DR. JUAN A. CEBRIAN
UNIVERSITY OF MARYLAND
DEPARTMENT OF GEOGRAPHY
COLLEGE PARK, MD 20742
(301) 454-2242

S. CHANDRAN
UNIVERSITY OF MARYLAND
COLLEGE PARK, MD 20742

DR. Y. PAUL CHIANG
WASHINGTON STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL
 AND COMPUTER ENGINEERING
PULLMAN, WA 99164-2210
(509) 335-2448

DR. MIKE LIN CHIEN-HUA
CLEVELAND STATE UNIVERSITY
COMPUTER AND INFORMATION SCIENCE
CLEVELAND, OH 44115
(216) 687-4780

MR. JOHN CHLUDZINSKI
INSTITUTE FOR DEFENSE ANALYSIS
1801 NORTH BEAUREGARD STREET
ALEXANDRIA, VA 22311
(703) 824-5518

SEOG YEON CHO
UNIVERSITY OF IOWA
123 FCB
IOWA CITY, IA 52242

MS. LESLIE J. CHOW
NASA-AMES RESEARCH CENTER
MAIL STOP 233-1
MOFFETT FIELD, CA 94035
(415) 694-6535

MR. DICK CLAYTON
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 021421
(617) 876-1111

MR. JAMES CLOSS
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-7877

MR. DAVID COHEN
UNIVERSITY OF IOWA
DEPARTMENT OF COMPUTER SCIENCE
101 MLH
IOWA CITY, IA 52240
(319) 354-4643

MR. ROBERT E. COOPER
LAWRENCE LIVERMORE NATIONAL LABORATORY
P.O. BOX 808
ATTN: L-16
LIVERMORE, CA 944550
(415) 422-4653

MR. TIMOTHY F. CROSSMAN
LOATS ASSOCIATES, INC.
24 NORTH COURT STREET
WESTMINSTER, MD 21157
(301) 876-8055

MR. KENT K. CURTIS
NATIONAL SCIENCE FOUNDATION
1800 G. STREET, NW
WASHINGTON, DC 20550
(202) 357-9747

MR. ERNEST DADDIO
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5209

DR. JILL P. DAHLBURG
1202 SOUTH WASHINGTON STREET
APARTMENT 516
ALEXANDRIA, VA 22314
(703) 549-2571

DR. RUSSELL B. DAHLBURG
1202 SOUTH WASHINGTON STREET
APARTMENT 516
ALEXANDRIA, VA 22314
(703) 549-2571

MR. MEL H. DAVIS
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-1282

DR. EDWARD W. DAVIS
NORTH CAROLINA STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
BOX 8206
RALEIGH, NC 27695
(919) 737-7045

DR. HOWARD B. DEMUTH
UNIVERSITY OF IDAHO
DEPARTMENT OF ELECTRICAL ENGINEERING
MOSCOW, ID 83843
(208) 885-7561

MS. JUDY DEVANEY
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-3021

MR. KAHAI DOLINH
INTEGRAL SYSTEMS, INC.
9701 S. PHILADELPHIA COURT
LANHAM, MD 20706
(301) 731-4233

DR. CLARK S. DONLEY
THE COLLEGIATE SCHOOLS
NORTH MOORELAND ROAD
RICHMOND, VA 23229
(804) 740-7077

DR. JOHN E. DORBAND
NASA-GODDARD SPACE FLIGHT CENTER
CODE 635
GREENBELT, MD 20771
(301) 286-9419

DR. JESSE M. DRAPPER
NATIONAL BUREAU OF STANDARDS
BUILDING 223/B364
GAITHERSBURG, MD 20899
(301) 921-3274

DR. NICHOLAS DUCHON
MARTIN MARIETTA LABS
1450 SOUTH ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

MR. TOM DUGAN
STI, INC.
NASA-GODDARD SPACE FLIGHT CENTER
CODE 401.1
GREENBELT, MD 20771
(301) 286-2882

DR. MICHAEL J. DUGGIN
STATE UNIVERSITY OF NEW YORK
NSAF
308 BRAY HALL, CESF
SYRACUSE, NY 13210
(315) 470-6701

DR. JAMES A. EARL
UNIVERSITY OF MARYLAND
DEPARTMENT OF PHYSICS AND ASTRONOMY
COLLEGE PARK, MD 20742
(301) 454-3136

MR. JIM EDELMANN
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5204

MR. TONE ENGEL
BROWN UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
BOX 1910
PROVIDENCE, RI 02912
(401) 863-1831

DR. MONNDY A. ESHERA
MARTIN MARIETTA LABS
1450 SOUTH ROLLING ROAD
BALTIMORE, MD 21227-3898
(301) 247-0700

MR. JOHN R. FAIRFIELD
JAMES MADISON UNIVERSITY
HARRISONBURG, VA 22807
(703) 568-6955

MR. RUDI O. FAISS
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-9888

MS. AI CHUN FANG
NASA HEADQUARTERS
CODE EI
WASHINGTON, DC 20546
(202) 453-1693

MR. HENRY C. FERGUSON
JOHNS HOPKINS UNIVERSITY
DEPARTMENT OF PHYSICS AND ASTRONOMY
ROWLAND HALL, 3400 N. CHARLES STREET
BALTIMORE, MD 21218
(301) 338-8698

DR. CHARLES FIDUCCIA
GENERAL ELECTRIC
RESEARCH AND DEVELOPMENT CENTER
SCHENECTADY, NY 12301
(518) 387-6583

DR. ROLF D. FIEBRICH
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MR. JAMES R. FISCHER
NASA-GODDARD SPACE FLIGHT CENTER
CODE 635
GREENBELT, MD 20771
(301) 286-3464

MR. BRETT FISHBURNE
WESTINGHOUSE
MARYLAND CORPORATE CENTER, SUITE 214
7301 FORBES BOULEVARD
SEABROOK, MD 20706
(301) 262-8868

DR. RICHARD P. FLANAGAN
ODSI DEFENSE SYSTEMS, INC.
6110 EXECUTIVE BOULEVARD, STE.320
ROCKVILLE, MD 20852
(301) 984-6006

DR. CHUNG B. FONG
MARTIN MARIETTA LAB
1450 S. ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

DR. DAVID W. FOX
UNIVERSITY OF MINNESOTA
DEPARTMENT OF COMPUTER SCIENCE
207 CHURCH STREET, S.E.
MINNEAPOLIS, MN 55455
(612) 625-0726

MS. KAREN A. FRENKEL
COMMUNICATIONS OF THE ACM
11 WEST 42ND STREET
NEW YORK, NY 10036
(212) 869-7440

MR. JACK C. FRITZ
IBM CORPORATION
15300 SHADY GROVE ROAD
ROCKVILLE, MD 20850
(301) 921-7532

MR. JAMES P. GARY
NASA-GODDARD SPACE FLIGHT CENTER
CODE 630.4
GREENBELT, MD 20771
(301) 286-9539

DR. JAMES A. GATLIN
NASA-GODDARD SPACE FLIGHT CENTER
CODE 725
GREENBELT, MD 20771
(301) 286-6680

DR. RON GHOSH
DARTMOUTH COLLEGE
HB-6127
HANOVER, NH 03755
(603) 646-3969

MR. PAUL GIORDANO
CONVEX COMPUTER CORPORATION
7474 GREENWAY CENTER DRIVE, 450
GREENBELT, MD 20770
(301) 345-2400

DR. IRVY GLEDHILL
STANFORD UNIVERSITY
STAR LABORATORY
337 DURAND BUILDING
STANFORD, CA 94305
(415) 723-3467

MR. ROBERT P. GORMAN
BENDIX AEROSPACE TECHNOLOGY CENTER
9140 OLD ANNAPOLIS ROAD
COLUMBIA, MD 21045
(301) 964-4094

DR. KIM P. GOSTELOW
GENERAL ELECTRIC RESEARCH CENTER
P.O. BOX 8
ROOM KWC-286
SCHENECTADY, NY 12308
(518) 387-5805

MR. MICHAEL L. GOUGH
SCIENCE APPLICATIONS RESEARCH
NASA-GODDARD SPACE FLIGHT CENTER
CODE 634
GREENBELT, MD 20771
(301) 286-9884

MR. JOHN G. GREGORY
WESTINGHOUSE
BOX 312
WHITE MARSH, MD 21162
(301) 765-4235

R. C. GROOME
BENDIX AEROSPACE
9140 OLD ANNAPOLIS ROAD
COLUMBIA, MD 21045
(301) 964-4094

DR. ROBERT J. GURNEY
NASA-GODDARD SPACE FLIGHT CENTER
CODE 624
GREENBELT, MD 20771
(301) 286-5480

MR. KURT M. GUTZMANN
PLANNING RESEARCH CORPORATION
1500 PLANNING RESEARCH DRIVE
MCLEAN, VA 22102
(703) 556-1529

MR. JAMES J. HACK
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
P.O. BOX 3000
BOULDER, CO 80307
(303) 497-1387

DR. MILTON HALEM
NASA-GODDARD SPACE FLIGHT CENTER
SPACE DATA AND COMPUTING DIVISION
CODE 630
GREENBELT, MD 20771
(301) 286-8834

DR. SUSANNE E. HAMBRUSCH
PURDUE UNIVERSITY
216 COMPUTER SCIENCE BUILDING
WEST LAYFAYETTE, IN 47907
(317) 494-1831

MS. SHERYL HANDLER
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MR. JONATHAN P. HARRIS
DIGITAL EQUIPMENT CORORATION
77 REED ROAD
HUDSON, MA 01749
(617) 568-6993

DR. ALBERT B. HARVEY
NATIONAL SCIENCE FOUNDATION
CISE/OASC, ROOM 533
1800 G STREET, N.W.
WASHINGTON, DC 20550
(202) 357-7727

MR. BAMBANG HARYMURTI
TIME MAGAZINE
1050 CONNECTICUT AVENUE, NW
WASHINGTON, DC 20036
(202) 861-4000

DR. HAROLD M. HASTINGS
HOFSTRA UNIVERSITY
MATH DEPARTMENT
HEMPSTEAD, NY 11530
(516) 560-5571

MR. WILLIAM HAYNES
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-3020

DR. SARA RIDGEWAY HEAP
NASA-GODDARD SPACE FLIGHT CENTER
ASTRONOMY BRANCH
CODE 681
GREENBELT, MD 20771
(301) 286-5359

MR. ROBERT S. HILL
STI CORPORATION
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-3624

DR. JESSE K. HILL
SASC TECHNOLOGIES
GODDARD SPACE FLIGHT CENTER
CODE 681
GREENBELT, MD 20771
(301) 286-7615

MR. DANNY HILLIS
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MR. ALLAN E. HINKLE
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-6361

PING HO
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5193

MR. DOUGLAS L. HOGAN
SPARTA, INC.
7926 JONES BRANCH DRIVE
SUITE 1070
MCLEAN, VA 22102
(703) 448-0210

MR. STEVE HOWELL
SASC TECHNOLOGIES
NASA-GODDARD SPACE FLIGHT CENTER
GREENBELT, MD 20771
(301) 286-3625

DR. CHRIS L. HOXIE
NUS CORPORATION
910 CLOPPER ROAD
GAITHERSBURG, MD 20878
(301) 258-8771

MRS. R. CAROL HOXIE
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-4097

DR. WEN-JING HSU
MICHIGAN STATE UNIVERSITY
702 CHERRY LANE
APARTMENT 107
EAST LANSING, MI 48823
(517) 353-3542

C. T. HUCKELBERRY
GOODYEAR AEROSPACE CORPORATION
1828 L STREET, NW
SUITE 801
WASHINGTON, DC 20036
(202) 331-7500

MR. YUBIN HUNG
UNIVERSITY OF MARYLAND
CENTER FOR AUTOMATION RESEARCH
COLLEGE PARK, MD 20742

MR. JAMES R. HURST
MRJ, INC.
10455 WHITE GRANITE DRIVE
OAKTON, VA 22124
(703) 385-0859

DR. NATHAN IDA
THE UNIVERSITY OF AKRON
ELECTRICAL ENGINEERING DEPARTMENT
ROOM 454
AKRON, OH 44325
(216) 375-7679

DR. KENNETH W. IOBST
SUPERCOMPUTING RESEARCH CENTER
4380 FORBES BOULEVARD
LANHAM, MD 20706
(301) 731-3728

DR. RICHARD B. ISSACMAN
APPLIED RESEARCH CORPORATION
8201 CORPORATE DRIVE
LANDOVER, MD 20785
(301) 286-5758

MS. KAREN JORDAN
TRW
ONE SPACE PARK
SPACE SCIENCE DEPARTMENT
REDONDO BEACH, CA 90278
(213) 536-2017

MR. BREWSTER KAHLE
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

DR. MICHAEL J. KASCIC, JR.
SUPERCOMPUTING RESEARCH CENTER
4380 FORBES BOULEVARD
LANHAM, MD 20706
(301) 731-3732

DR. ASHOK G. KAVEESHWAR
SASC TECHNOLOGIES
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5010

MR. JOHN KELLER
ADVANCED MILITARY COMPUTING
1401 WILSON BOULEVARD
SUITE 910
ARLINGTON, VA 22209
(703) 528-1244

MR. BERNARD LEE KILDAY
SPARTA, INC.
23293 SO. POINTE DRIVE
LAGUNA HILLS, CA 92653
(714) 768-8161

MR. LARRY KLEIN
APPLIED RESEARCH CORPORATION
8201 CORPORATE DRIVE
LANDOVER, MD 20785
(301) 286-2595

MR. BEN KOBLER
MARTIN MARIETTA LABS
1450 SOUTH ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

MR. TOM KRAAY
MRJ, INC.
10455 WHITE GRANITE DRIVE
OAKTON, VA 22124
(703) 385-0859

MR. HAL S. KUFF
BOX 168
STEVENSON, MD 21153
(301) 494-0086

DR. ROBERT CHARLES LADNER
GENEX CORPORATION
16020 INDUSTRIAL DRIVE
GAITHERSBURG, MD 20877
(301) 258-0552

MR. PETER G. LANG
MCDONNELL DOUGLAS CORPORATION
8201 GREENSBORO DRIVE
MCLEAN, VA 22102
(703) 883-4901

DR. KATHRYN B. LASKEY
DECISION SCIENCE CONSORTIUM
7700 LEESBURG PIKE
FALLS CHURCH, VA 22043
(703) 790-0510

DR. HARRY C. LEE
MARTIN MARIETTA AEROSPACE
P.O. BOX 5837
MAIL POINT 483
ORLANDO, FL 32855
(305) 356-4233

DR. JOHN W. LEWIS
MARTIN MARIETTA LABORATORIES
1450 S. ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

MR. AARON J. LEWIS
US PATENT AND TRADEMARK OFFICE
P.O. BOX 2251
ARLINGTON, VA 22202
(703) 557-4916

DR. CHIN S. LIN
SOUTHWEST RESEARCH INSTITUTE
DEPARTMENT OF SPACE SCIENCES
6220 CULEBRA ROAD
SAN ANTONIO, TX 78284
(512) 522-3541

MR. PETER E. LOATS
LOATS ASSOCIATE, INC.
24 NORTH COURT STREET
WESTMINSTER, MD 21157
(301) 876-8055

MR. HARRY L. LOATS
LOATS ASSOCIATES
24 NORTH COURT STREET
WESTMINSTER, MD 21157
(301) 876-8055

MR. RON G. LOELIOGER
MACAULAY-BROWN, INC.
3915 GERMANY LANE
DAYTON, OH 45431
(513) 426-3421

MR. DICK LOTT
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
DEPARTMENT 470/C2E
AKRON, OH 44315
(216) 796-2457

MR. DENNIS M. LYNCH
PRC, INC.
3030 ARROWHEAD DRIVE
FAIRFAX, VA 22030
(703) 385-6880

DR. GORDON E. LYON
NATIONAL BUREAU OF STANDARDS
PARALLEL PROCESSING GROUP
B-364 MATERIALS BUILDING
GAITHERSBURG, MD 20899
(301) 921-3274

DR. STEPHEN MANGO
NAVAL RESEARCH LABORATORY
CODE 5381MA
WASHINGTON, DC 20375-5000
(202) 767-2003

MR. FRANK J. MARSHALL
CONVEX COMPUTER CORPORATION
701 PLANO ROAD
RICHARDSON, TX 75081
(214) 952-0289

DR. WILLIAM MATTHAEUS
BARTOL RESEARCH FOUNDATION
UNIVERSITY OF DELAWARE
NEWARK, DE 19716
(302) 451-8111

DR. ROBERT M. MATTHEYSES
GENERAL ELECTRICAL RESEARCH
AND DEVELOPMENT
P.O. BOX 8 (KWC 288)
SCHENECTADY, NY 12301
(518) 387-6581

PROF. DOUGLAS MAURER
GEORGE WASHINGTON UNIVERSITY
(SEAS)
WASHINGTON, DC 20052
(202) 676-6569

DR. MICHAEL MCANULTY
UNIVERSITY OF ALABAMA IN BIRMINGHAM
DEPARTMENT OF COMPUTER AND INFORMATION
SCIENCE
BIRMINGHAM, AL 35294
(205) 934-2213

DR. DUDLEY G. MCCONNELL
NASA HEADQUARTERS
CODE E
WASHINGTON, DC 20546
(202) 453-1420

DR. CALDWELL MCCOY
NASA HEADQUARTERS
CODE EI
WASHINGTON, DC 20546
(202) 453-1495

MS. KATHY J. MCCURDY
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-7265

MS. EMMA MCKNIGHT
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-6034

DR. PAUL R. MCMULLIN
JOHNS HOPKINS APPLIED PHYSICS LAB
JOHNS HOPKINS ROAD
LAUREL, MD 20707
(301) 953-5000

MR. JAMES F. MCSHANE
1072 JAMES STREET
APARTMENT C-17
SYRACUSE, NY 13203
(315) 456-6505

MR. BRAD MILLER
DIGITAL EQUIPMENT CORPORATION
77 REED ROAD
HUDSON, MA 01749
(617) 568-4741

DR. RAYMOND D. MOUNTAIN
NATIONAL BUREAU OF STANDARDS
A311 PHYSICS
GAITHERSBURG, MD 20899
(301) 921-2831

DR. A. P. MULLHAUPT
UNIVERSITY OF NEW MEXICO
DEPARTMENT OF MATHEMATICS
ALBUQUERQUE, NM 87131
(505) 277-3340

MR. JERRY NARDI
TECHNO SYSTEMS, INC.
7815-203 MANDAN ROAD
GREENBELT, MD 20770
(301) 286-8890

PROF. DAVID NASSIMI
UNIVERSITY OF DELAWARE
CIS DEPARTMENT
103 SMITH HALL
NEWARK, DE 19716
(302) 451-1959

MR. E. A. NEIL
NASA-GODDARD SPACE FLIGHT CENTER
SPACE DATA AND COMPUTING DIVISION
CODE 630
GREENBELT, MD 20771
(301) 286-6989

MR. ROGER NEILSEN
AMT, INC.
4 WAYSIDE LANE
ACTON, MA 01720
(617) 263-2333

MR. BRUCE J. NEMNICH
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

DR. MASAHIRO NISHIHAMA
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-5207

MR. BRUCE NORDWALL
AVIATION WEEK AND SPACE TECHNOLOGY
1777 N. KENT STREET
SUITE 710
ARLINGTON, VA 22209
(703) 875-2842

DR. JAY P. NORRIS
NAVAL RESEARCH LABORATORY
CODE [illegible]
WASHINGTON, DC 20375-5000
(202) 767-2781

MS. [illegible] A. O'CONNELL
817 [illegible] ROAD
ROCKVILLE, MD 20852
(301) 251-1738

DR. JOHN O'DONNELL
INDIANA UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
BLOOMINGTON, IN 47405
(812) 335-0739

MR. CARLOS OJEDA
NASA HEADQUARTERS
NOAA SATELLITE RESEARCH LAB
2205 MOUNT TABOR ROAD
GAMBRILLS, MD 21054
(301) 763-4240

MR. DAN E. OLDFIELD
INTERNATIONAL COMPUTING LTD.
ESKDALE ROAD, WINNERSH
WOKINGHAM, BERKSHIRE RG115TT UK
+44-734-693131

MR. DAVID W. OLIVER
GENERAL ELECTRIC COMPANY
P.O. BOX 8
BUILDING K-1, ROOM 5C17
SCHENECTADY, NY 12301
(518) 387-5039

DR. TORSTEIN OPSAHL
MRJ, INC.
PERKIN-ELMER ADVANCED DEVELOPMENT CENTER
10467 WHITE GRANITE DRIVE
OAKTON, VA 22124
(703) 246-9617

MR. MARTIN OZGA
U.S. DEPARTMENT OF AGRICULTURE-NASS
ROOM 4842 SOUTH AGRICULTURE BUILDING
WASHINGTON, DC 20250
(202) 447-5483

DR. JO ANN PARIKH
SOUTHERN CONNECTICUT STATE UNIVERSITY
877 MOUNT CARMEL AVENUE
NORTH HAVEN, CT 06473
(302) 397-4514

PROF. DENNIS PARKINSON
DAP SUPPORT UNIT
QUEEN MARY COLLEGE
LONDON, ENGLAND
011-1-980-4811

DR. JOHN S. PERKO
NASA-GODDARD SPACE FLIGHT CENTER
CODE 665
GREENBELT, MD 20771
(301) 286-4275

MR. IVARS PETERSON
SCIENCE NEWS
1719 N STREET, N.W.
WASHINGTON, DC 20036
(202) 785-2255

DR. FRANK M. PITELLI
U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
ANNAPOLIS, MD 21402-5002
(301) 267-2679

DR. SANJAY POL
NORTH CAROLINA STATE UNIVERSITY
DEPARTMENT OF COMPUTER STUDIES
RALEIGH, NC 27606
(919) 828-3407

MS. LAURA A. POTLER
NASA-GODDARD SPACE FLIGHT CENTER
CODE 635
GREENBELT, MD 20771
(301) 286-9420

DR. JERRY L. POTTER
KENT STATE UNIVERSITY
MATH SCIENCES DEPARTMENT
KENT, OH 44242
(216) 672-2560

MS. BARBARA H. PUTNEY
NASA-GODDARD SPACE FLIGHT CENTER
CODE 621
GREENBELT, MD 20771
(301) 286-6018

MR. GARY RANCOURT
THINKING MACHINES CORPORATION
245 FIRST STREET
CAMBRIDGE, MA 02142
(617) 876-1111

MS. ANNE C. RAUGH
SASC TECHNOLOGIES
NASA-GODDARD SPACE FLIGHT CENTER
CODE 681
GREENBELT, MD 20771
(301) 286-3625

PROF. ANTHONY P. REEVES
UNIVERSITY OF ILLINOIS
319L TALBOT LABORATORY
104 SOUTH WRIGHT STREET
URBANA, IL 61801
(217) 244-5978

DR. JOHN REIF
DUKE UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
DURHAM, NC 27706
(919) 684-3048

DR. JOHN P. RIGANATTI
SUPERCOMPUTING RESEARCH CENTER
4380 FORBES BOULEVARD
LANHAM, MD 20706
(301) 731-3741

MR. GARY RILEY
NASA-JOHNSON SPACE CENTER
FM7
HOUSTON, TX 77058
(713) 483-4751

MR. JOHN W. ROBERTS
NATIONAL BUREAU OF STANDARDS
B364 MATLS
GAITHERSBURG, MD 21701
(301) 921-3274

MS. NANCY E. RODGER
303 TENNESSEE AVENUE, N.E.
WASHINGTON, DC 20002
(301) 288-4357

DR. JEFFREY D. ROSENDAHL
NASA HEADQUARTERS
CODE E
WASHINGTON, DC 20546
(202)453-1410

MR. GARY W. SABOT
HARVARD UNIVERSITY
287 HARVARD STREET #39
CAMBRIDGE, MA 02139
(617) 876-1111

DR. NARAYAN SAHOO
STATE UNIVERSITY OF NEW YORK AT ALBANY
DEPARTMENT OF PHYSICS
1400 WASHINGTON AVENUE
ALBANY, NY 12222
(518) 442-4549

MR. DOUGLAS M. SAKAL
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5208

DR. VINCENT SALOMONSON
NASA-GODDARD SPACE FLIGHT CENTER
CODE 620
GREENBELT, MD 20771
(301) 286-6481

DR. DAVID SALZMAN
NATIONAL SCIENCE FOUNDATION
1800 G STREET, NW
WASHINGTON, DC 20550
(202) 357-9776

MR. BOB SAVELY
NASA-JOHNSON SPACE CENTER
FM7
HOUSTON, TX 77058
(713) 483-4751

PROF. DAVID SCHAEFER
GEORGE MASON UNIVERSITY
DEPARTMENT OF ELECTRICAL &
COMPUTER ENGINEERING
FAIRFAX, VA 22030
(703) 323-2009

DR. ERWIN SCHMERLING
NASA HEADQUARTERS
CODE E
WASHINGTON, DC 20546
(202) 453-1459

MR. EDWARD J. SCHOLZ
ORION RESEARCH, INC.
1811 ABBEY OAK DRIVE
VIENNA, VA 22180
(703) 938-3482

MR. JOSEPH SEAMONE
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-7877

MR. RICHARD SEARS
412 WATAUGA DRIVE
APARTMENT 3
KNOXVILLE, TN
(615) 688-8204

MR. EDWARD SEILER
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-9421

MR. ERNEST SIBERT
SYRACUSE UNIVERSITY
313 LINK HALL
SYRACUSE, NY 13244-1240
(315) 423-2369

MR. DOUG R. SMITH
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-7263

DR. CARL H. SMITH
NATIONAL BUREAU OF STANDARDS
B364 MATLS
GAITHERSBURG, MD 20899
(301) 454-7930

MR. ROBERT D. SNELICK
NATIONAL BUREAU OF STANDARDS
B364 MATLS
GAITHERSBURG, MD 20899
(301) 921-3274

MR. MARK A. STEVENS
SCIENCE APPLICATIONS RESEARCH
CODE 630
NASA-GODDARD SPACE FLIGHT CENTER
GREENBELT, MD 20771
(301) 286-9415

DR. L.R. OWEN STOREY
STANFORD UNIVERSITY
333 DURAND BUILDING
STANFORD, CA 94305
(415) 723-9258

DR. JAMES P. STRONG
NASA-GODDARD SPACE FLIGHT CENTER
CODE 636
GREENBELT, MD 20771
(301) 286-9435

DR. FRANCIS SULLIVAN
NATIONAL BUREAU OF STANDARDS
SCIENTIFIC COMPUTING DIVISION
BUILDING 225, ROOM A151
GAITHERSBURG, MD 20899
(301) 921-3395

PROF. DANIEL TABAK
GEORGE MASON UNIVERSITY
DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
FAIRFAX, VA 22030
(703) 323-4143

MR. LARRY L. TAYLOR
GOODYEAR AEROSPACE CORPORATION
1210 MASSILLON ROAD
AKRON, OH 44315
(216) 796-2801

MR. ROBERT F. TEITEL
TEITEL DATA SYSTEMS
SUITE 727 EAST
7315 WISCONSIN AVENUE
BETHESDA, MD 20814
(301) 656-0401

MR. JOHN R. THORP
MARTIN MARIETTA LABS
1450 S. ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

MR. ALYSON L. THRING
SOUTHWEST RESEARCH INSTITUTE
DEPARTMENT OF SPACE SCIENCES
6220 CULEBRA ROAD
SAN ANTONIO, TX 78284
(512) 684-5111

DR. JAMES C. TILTON
NASA-GODDARD SPACE FLIGHT CENTER
CODE 636
GREENBELT, MD 20771
(301) 286-9510

MR. LLOYD TREINISH
NASA-GODDARD SPACE FLIGHT CENTER
CODE 634
GREENBELT, MD 20771
(301) 286-9884

DR. YEDATORE V. VENKATESH
NASA-GODDARD SPACE FLIGHT CENTER
CODE 636
GREENBELT, MD 20771
(301) 286-4094

MR. WILLIAM VON DER HAAR
CASE WESTERN RESERVE UNIVERSITY
CLEVELAND, OH 44106
(216) 368-4020

DR. SCOTT A. VON LAVEN
KMS FUSION, INC.
P.O. BOX 1567
ANN ARBOR, MI 48106
(313) 769-8500

MR. MITCHELL S. WADE
DIGITAL PRODUCTIONS
3416 S. LA CIENEGA BOULEVARD
LOS ANGELES, CA 90016
(213) 938-1111

MR. DONALD P. WALSH
US PATENT & TRADEMARK OFFICE
P.O. BOX 2231
ARLINGTON, VA 22202
(703) 557-3981

DR. PAUL C. WANG
ITT AVIONICS
390 WASHINTON AVENUE
NUTLEY, NY 07110
(301) 284-3815

MR. ARCHIBALD WARNOCK
STI CORPORATION
NASA-GODDARD SPACE FLIGHT CENTER
CODE 681
GREENBELT, MD 20771
(301) 286-8968

DR. SCOTT T. WEIDMAN
MRJ, INC.
10455 WHITE GRANITE DRIVE
OAKTON, VA 22124
(703) 385-0781

DR. JANET L. WEILAND
APPLIED RESEARCH CORPORATION
8201 CORPORATE DRIVE
SUITE 920
LANDOVER, MD 20785
(301) 459-8442

MR. FRED WEINGARD
BOOZ-ALLEN & HAMILTON, INC.
CRYSTAL SQUARE 2, SUITE 1100
1725 JEFFERSON DAVIS HIGHWAY
ARLINGTON, VA 22202
(703) 553-3578

MAJOR GLEN R. WHITE
US ARMY CONCEPTS ANALYSIS AGENCY
(CSCA-RSA)
8120 WOODMONT AVENUE
BETHESDA, MD 20814-2797
(301) 295-0303

DR. RICHARD A. WHITE
APPLIED RESEARCH CORPORATION
8201 CORPORATE DRIVE
SUITE 920
LANDOVER, MD 20785
(301) 286-7802

DR. RICHARD L. WHITE
SPACE TELESCOPE SCIENCE INSTITUTE
HOMEWOOD CAMPUS
BALTIMORE, MD 21218
(301) 338-4797

MR. DAVID WILDENHAIN
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 286-3021

LT. ALAN N. WILLIAMS
U.S. AIR FORCE
ROME AIR DEVELOPMENT CENTER (USAF)
ROME, NY 13441
(315) 330-3681

MR. ARCHIE E. WILLIAMS, JR.
U.S. PATENT AND TRADEMARK OFFICE
P.O. BOX 2231
ARLINGTON, VA 22202
(703) 557-2119

MR. BRYAN WOLF
MCDONNELL DOUGLAS INC.
8201 GREENSBORO DRIVE
MCLEAN, VA 22102
(301) 670-5313

MS. JENNY WU
SCIENCE APPLICATIONS RESEARCH
4400 FORBES BOULEVARD
LANHAM, MD 20706
(301) 794-5145

DR. MARVIN WUNDERLICH
DEPARTMENT OF DEFENSE
FORT MEADE, MD
(301) 859-6461

DR. PEN-SHU YEH
MARTIN MARIETTA LABS
1450 SOUTH ROLLING ROAD
BALTIMORE, MD 21227
(301) 247-0700

DR. LO I. YIN
NASA-GODDARD SPACE FLIGHT CENTER
CODE 682
GREENBELT, MD 20771
(301) 286-5682

ILLUSTRATIONS

PRECEDING PAGE BLANK NOT FILMED

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 4

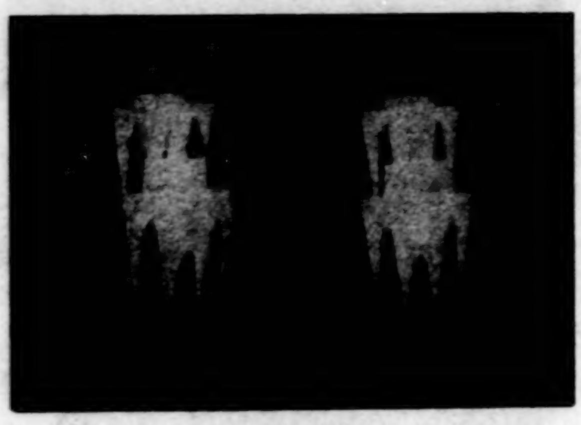


Figure 5

Figure 6

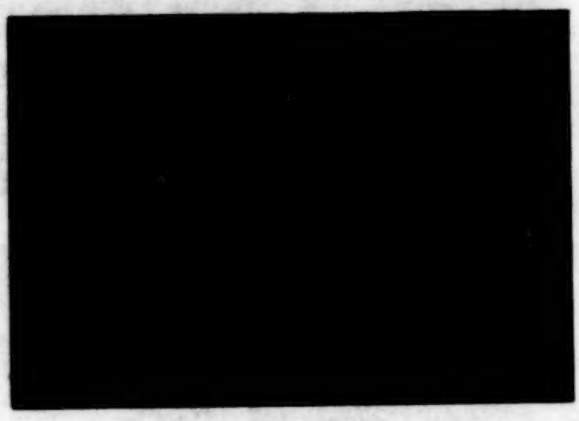


Figure 7

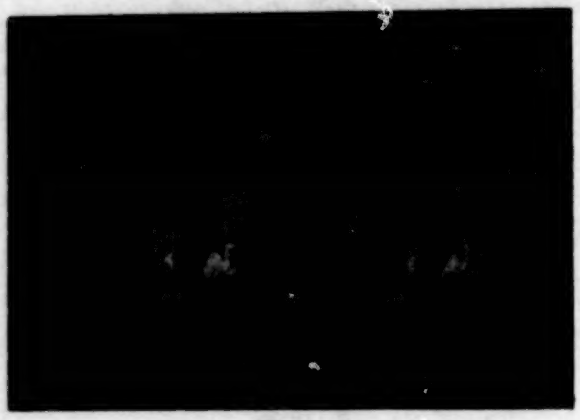


Figure 8

Figure 9

Plate I. Stereo Images of Digital Terrain

PRECEDING PAGE BLANK NOT FILMED

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 1

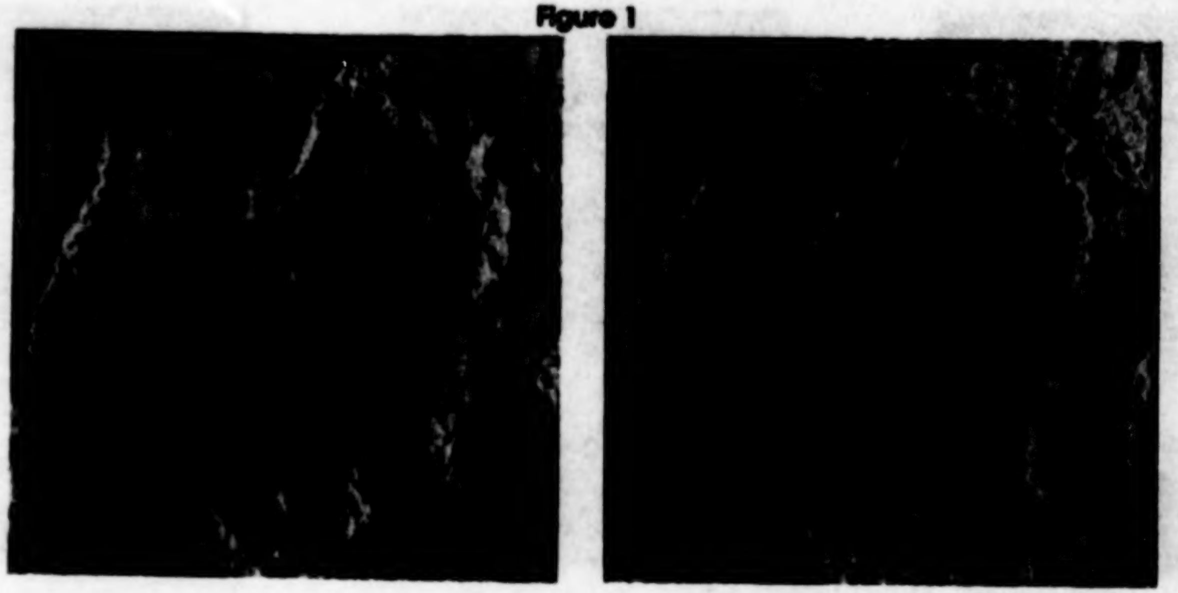


Figure 2

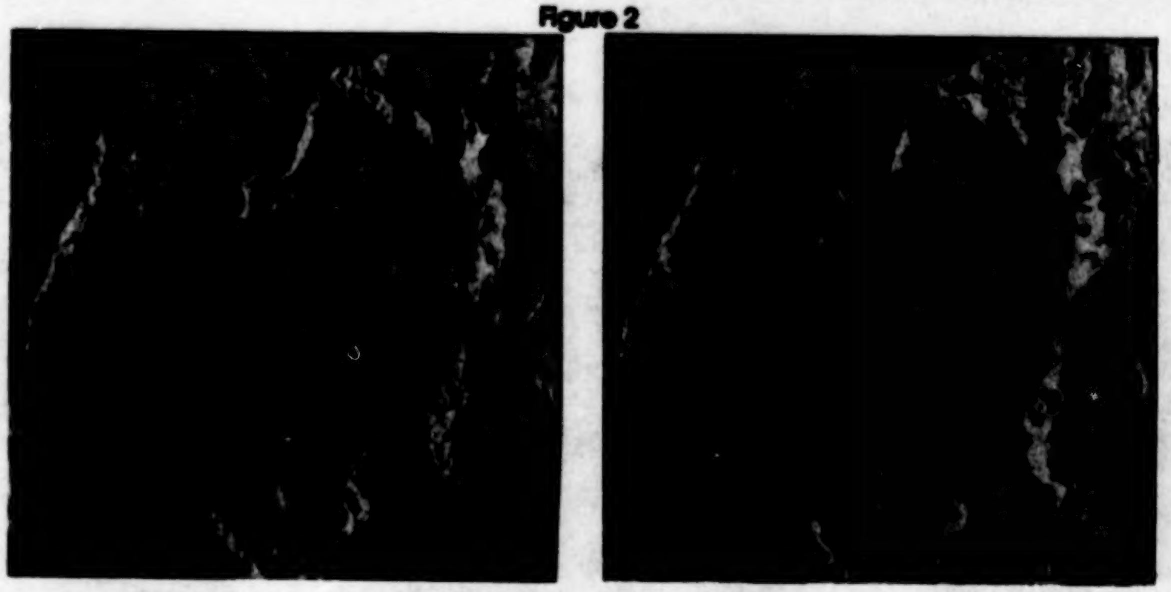


Plate II. Stereo Analysis Algorithms for Generating Topological Maps I

Figure 1: Stereo Pair of SAR Images of Plateau Region Along India-Bangladesh Border.
Figure 2: Right Image Warped After Two Iterations of Matching Algorithm.

ORIGINAL PAGE IS
OF POOR QUALITY

Plate III. Stereo Analysis Algorithms for Generating Topological Maps II

ORIGINAL PAGE IS
OF POOR QUALITY

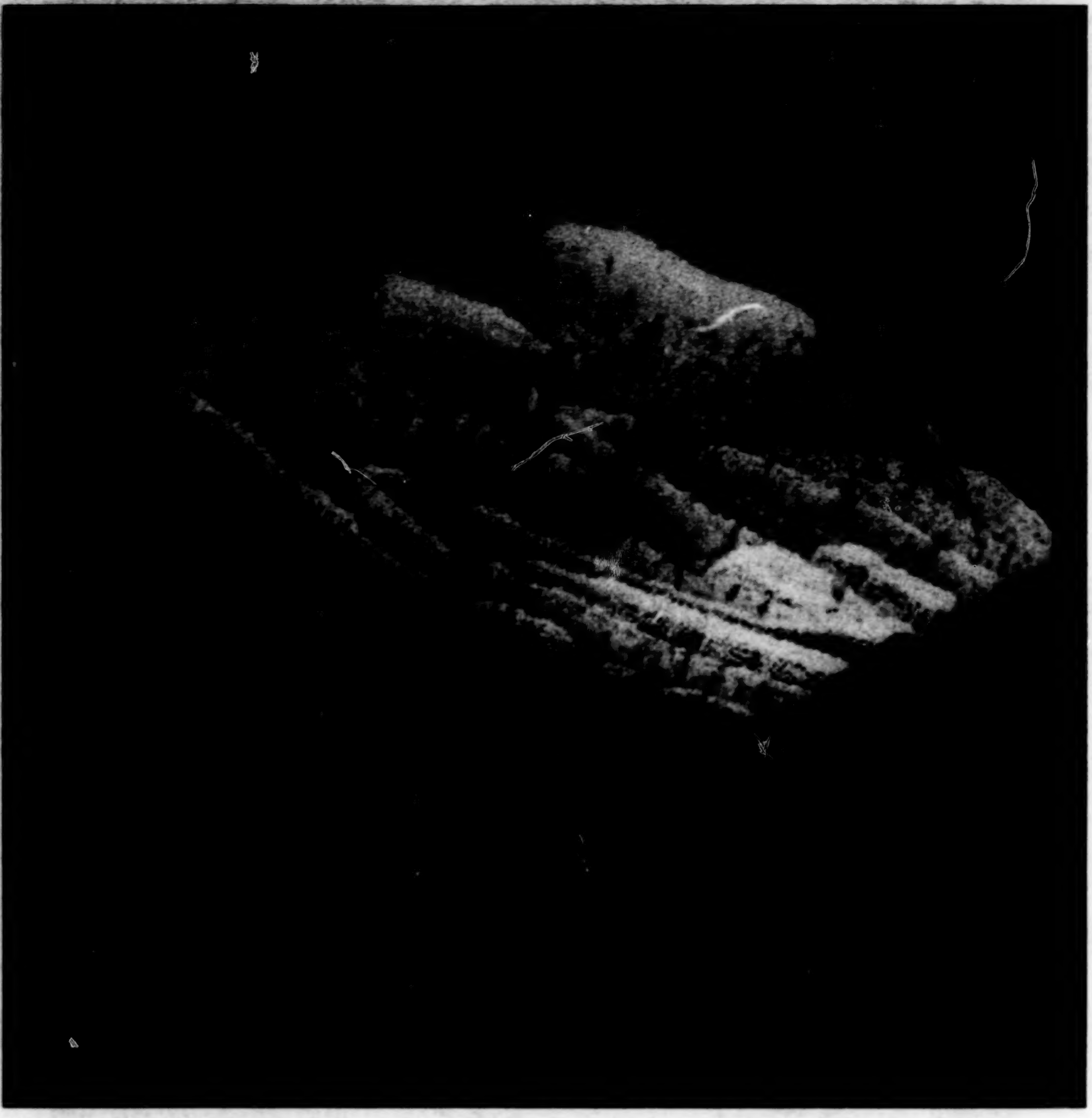


Plate IV. Ray Tracing on the MPP

A 3-D Rendering of an Elevation Map of the Susquehanna and Julianna Rivers in the Harrisburg., Pennsylvania Area

ORIGINAL PAGE IS
OF POOR QUALITY

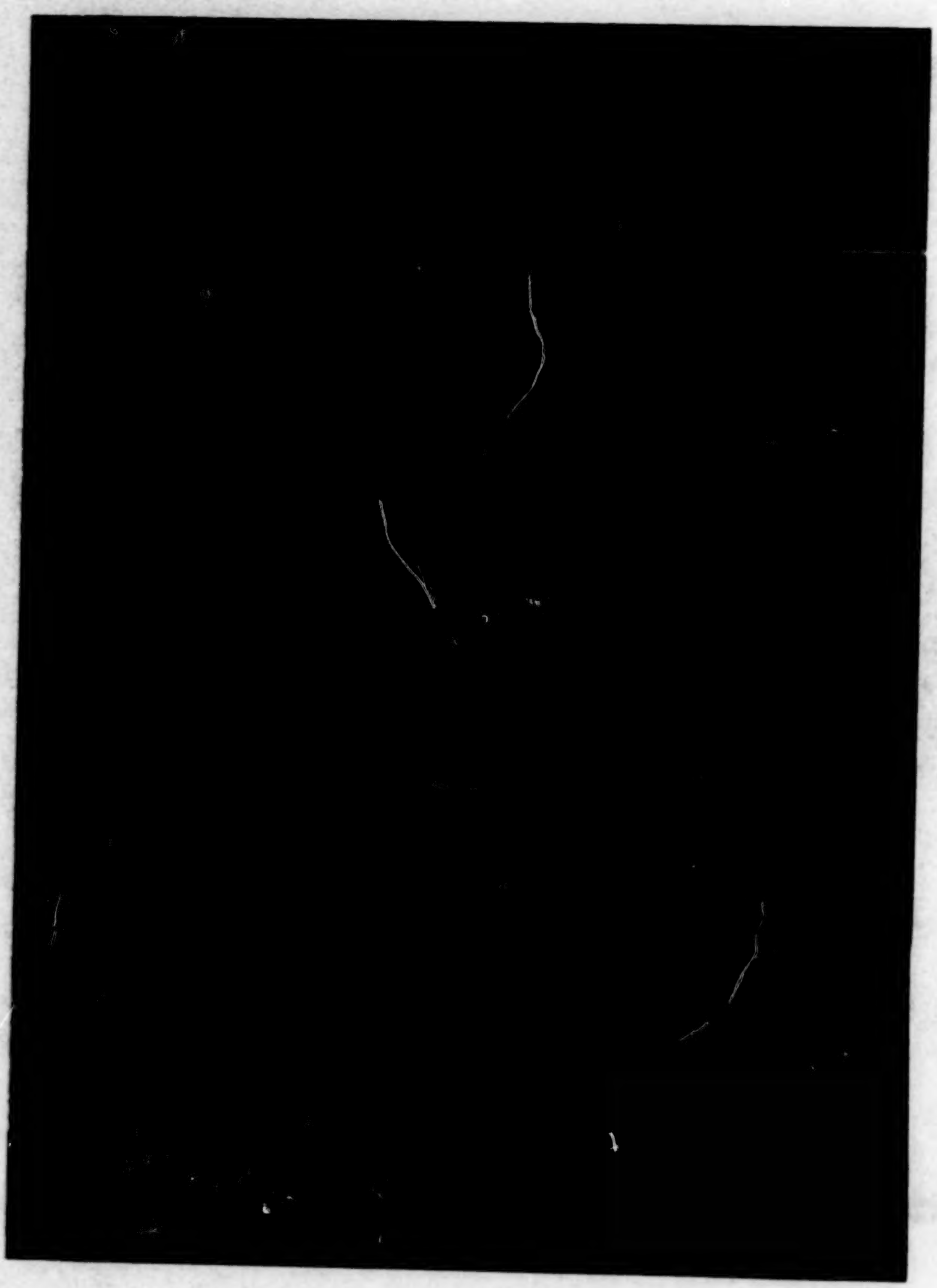


Plate V. Animated Computer Graphics Models of Space and Earth Sciences Data

Spatial and Spectral Distribution of Gridded Energy Fluxes Calculated from Remotely Sensed X-Ray Counts Observed from a Sounding Rocket Instrument During an Auroral Event.

ORIGINAL PAGE IS
OF POOR QUALITY

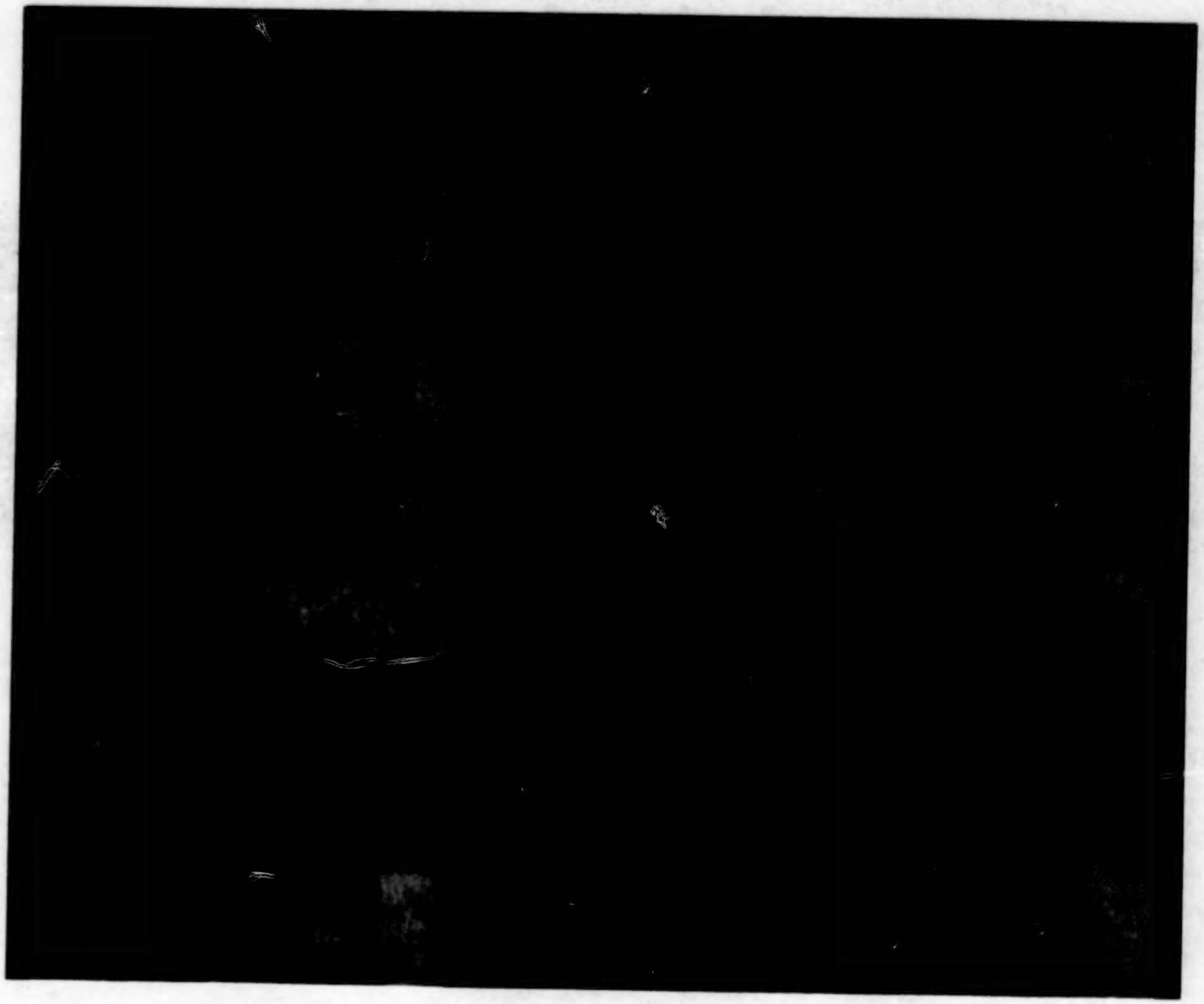


Plate VI. Algorithmic Commonalities in the Parallel Environment I

Evolution of a Piece of Sky Created by Stochastic Interpolation. Left-to-Right, Top-to-Bottom Shows Progression of the Subdivisions from Level 1 Through Level 4.

ORIGINAL PAGE IS
OF POOR QUALITY

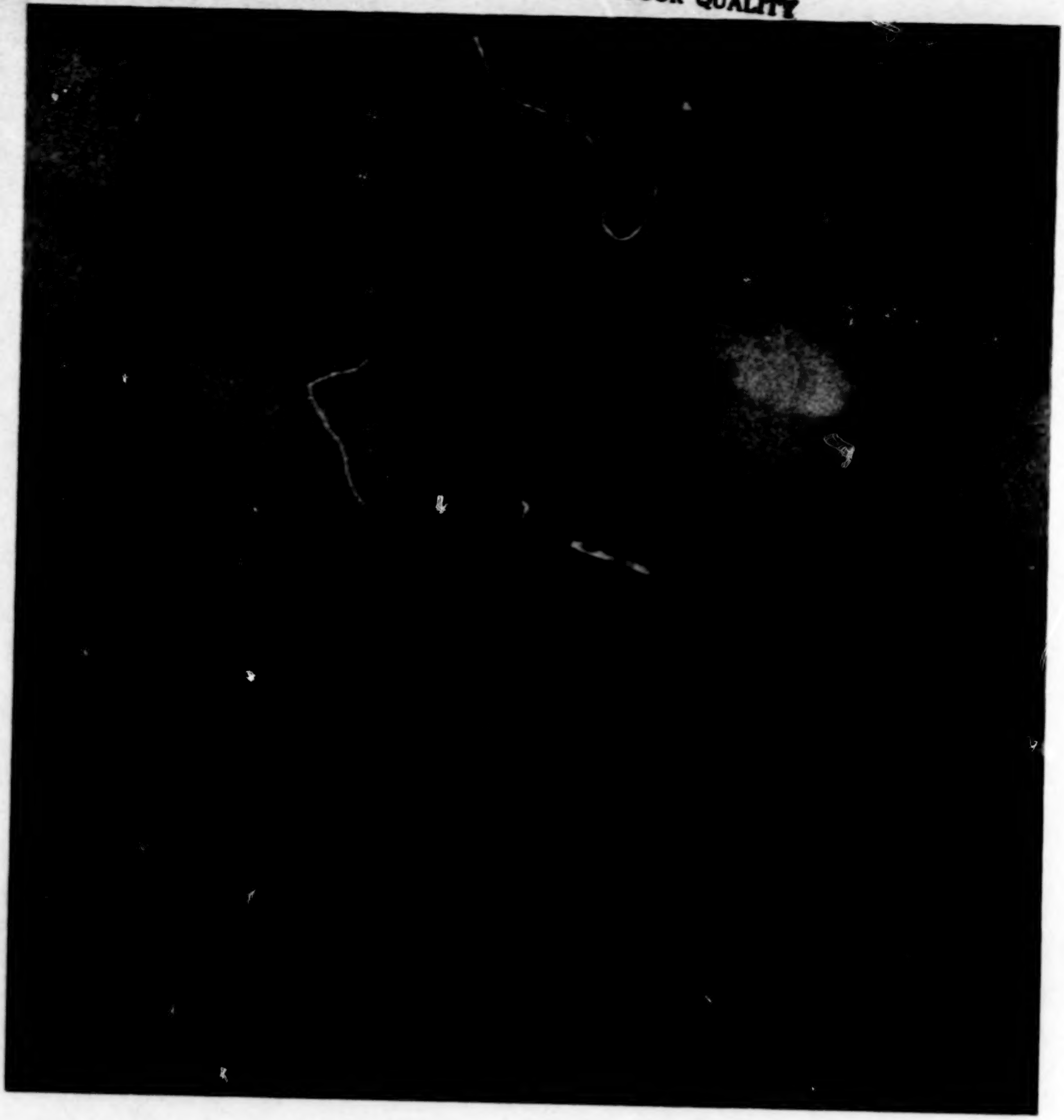


Plate VII. Algorithmic Commonalities in the parallel Environment II

Composite Image Combining Stochastically Intepolated Patches of Different h Factor, Scale, and Color Maps

ORIGINAL PAGE IS
OF POOR QUALITY

PLATE VIII: THE MPP WORKING GROUP

Front row (l-r): Dr. Milton Halem, Dr. James P. Strong, Dr. Scott Von Laven, Paul Harten, Dr. Irvy M. A. Gledhill, Dr. Nathan Ida, Karen F. Jordan, and Dr. Chin S. Lin. **Second row:** Dr. James C. Tilton, Dr. James A. Earl, Seog Y. Cho, Martin Ozga, Dr. Kenneth E. Batcher, and Dr. John T. O'Donnell. **Third row:** James R. Fischer, Edward J. Seiler, Dr. Richard L. White, Dr. Michael J. Bielefield, and Dr. Edward W. Davis.**Fourth row:** Dr. John E. Dorband, Dr. Marvin C. Wunderlich, Dr. Harold M. Hastings. **Back row:** Dr. Dennis Parkinson, Dr. Chester E. Grosch, Dr. John A. Barnden, Dr. Michael A. McAnulty, and Lloyd A. Treinish.

PRECEDING PAGE BLANK NOT FILMED

PRECEDING PAGE BLANK NOT FILMED

END

DATE

FILMED

OCT 9 198